65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 7,-

4

DEZEMBER 1978

ELECTRONICA UND IPE

Auf der Messe vor zwei Jahren trafen die damals ausgestellten Grundsysteme auf ein noch wenig vorbereitetes Publikum. Dieses hat die seither stürmische Weiterentwicklung schnell mitvollzogen und interessierte sich bereits stärker für die höher ausgebauten Systeme und die dazu angebotene Peripherie.

Die parallel abgehaltene IPE zog stattliche 5564 Besucher nach München. Ein neues Publikum, mittelständische Anwender und interessierte Laien, prüfte dort das Angebot an funktionsfähig aufgebauten Systemen und versah sich mit geeigneter Literatur.

Bei den 8-Bit-Mikros darf man nach diesen zwei Jahren eine frühe Blüte feststellen. Leistungsfähige Monitorprogramme, Text-Editoren, Assembler und BASIC-Interpreter erleichtern dem Anwender die eigene Programmentwicklung und verbessern den Dialog Mensch-Maschine. Bei den 65xx-Mikros fanden daher der AIM 65 und der SYM-1 als Neuvorstellungen große Beachtung, nicht minder die MCS-Systeme, der PET und der APPLE.

Mit diesen Werkzeugen in der Hand darf sich der Benutzer immer weniger mit dem Maschinencode befassen und kann sich immer mehr mit der Lösung seines eigentlichen Problems beschäftigen. Die verbesserten Betriebssysteme und sicher auch die im 65xx MICRO MAG erscheinenden Programme werden immer mehr zu einem 'modularen Programmieren' führen, zum Heranziehen vor-

INHALTSVERZEICHNIS

Fortsetzung von Seite 1:

handener und bewährter Teillösungen zur Bearbeitung ganzer Datenfelder und Dateien. Das Denken in Opcodes, Maschinenadressen und Bytes tritt mehr und mehr zurück.

Bei den höheren Programmiersprachen werden zum BASIC weitere Interpreter treten aber auch sehr leistungsfähige Compilersprachen, die die Ausführungszeiten nachhaltig verbessern. Die vielen im In- und Ausland hierzu erkennbaren Ansätze lassen erwarten, daß bis zur nächsten ELECTRONICA ein gewisser Kulminationspunkt in der Softwareentwicklung erreicht sein wird. Nach der eingehenden Besprechung der neuen Systeme wird diese Zeitschrift daher zunehmend auf die Programmierung in den höheren Sprachen eingehen, zunächst natürlich auf BASIC.

Aber auch die Technologie bedingt für die Programmierung dauernde Veränderung und Optimierung. Die neuen Peripheriebausteine werden zusehends leistungsfähiger und entlasten die CPU nachhaltig. Hingewiesen sei z.B. auf die in Heft 3, S. 41 besprochenen 'intelligenten' CRT-Controller, Keyboard/Display-Controller etc.. Durch einfaches initialisierendes Schreiben in die zahlreichen Register dieser Bausteine werden sie in ihren digitalen Funktionen geschaltet und können dann eine recht unabhängige Eigenverwaltung ihrer Dienstleistung vornehmen. In Laufe der Zeit wird man mehr und mehr auch zeit- und rechenintensive Vollzüge an Tochtereinheiten delegieren (Number Chruncher u.ä.). Die CPU als Baustein mit der höheren Intelligenz gewinnt damit Zeit als Manager für noch mehr Peripherie und für mehr logische Entscheidungen. Der Bedarf für ein wirksames Interruptmanagement (Hardware) wird allerdings gleichzeitig steigen.

Die preiswerten 8-Bit-Mikros stehen damit noch vor einer Hauptblüte ihrer Anwendung, und schon werden die 16-Bit-CPUs angekündigt. Sie sind für größere Effektivität in der Programmierung und für die Beherrschung großer residenter Speicher ausgelegt. Sie zielen damit auf eine Leistungsklasse, die bisher wesentlich größeren Rechengeräten vorbehalten war. Und man darf erwarten, daß sie mit den aus den 8-Bit-Maschinen gewonnenen Erfahrungen in relativ kurzer Zeit zu Entwicklungssystemen und zu Anwendungen gelangen.

Für die 65xx-Systemfamilie liegt noch keine feste Ankündigung und Beschreibung eines 16-Bit-Prozessors vor. Sicher ist es nicht abwegig, für ihn eine ähnliche Auslegung zu erwarten, wie sie der MC68000 von Motorola hat. Und bei diesem sieht man doch hochinteressante Weiterentwicklungen:

Dazu gehören 64-Pin-Gehäuse, 23 Adreßpins (16 Megabyte adressierbar), 16-Bit-Datenbus, beide Busse ohne Multiplexing, 16 Stück Maschinenregister, alle als Indexregister verwendbar, 24-Bit-Programmzähler, 16-Bit-Statusregister mit Steuerung für 8 Interrupt-Prioritätsebenen, erweiterter Instruktionssatz u.a. mit Multiply und Divide, STM/LDM Store/Load Multiple Registers und Befehle zur Codewandlung ASCII to packed BCD und umgekehrt. 14 Adressierungsarten stehen zur Verfügung, die denen großer Maschinen sehr ähneln. So kann die effektive Adresse nicht nur direkt und aus den Registerinhalten direkt sondern auch indirekt ggfs. unter Addition eines Offset und eines Index gebildet werden. Dabei gibt es noch die Spielarten, daß der Registerinhalt entweder vor oder nach der Operation um 1,2 oder 4 erhöht wird (Rasterung von Tabellen). Weiterhin ist eine Adressierung relativ zum Programmzähler möglich, mit Offset oder mit Offset + Index.

Diese kurze Aufzählung läßt bereits erahnen, welche Möglichkeiten mit der nächsten Prozessorgeneration auf uns zukommen.

R.L.

Ein Leitfaden für die Programmierung

E: This series, A Guideline to Programming, first of all asks the programmer to be conscious about the target of his beginnings. Then to proceed systematically, analyzing the hardware surroundings of his application, the timing and logical conditions and to lay down his approach into a flowchart or block diagram of consecutive actions to be executed. Thereafter the useful structure of a program is discussed, leading over to the procedure of detail-planning and actually writing the program. Application of structured programming with subroutines is very recommended and the competitive usage of index registers by caller and by subroutine discussed in detail.

1. Einleitung

Seit einiger Zeit war für diese Zeitschrift ein Grundsatzartikel für die Programmierung angekündigt. Wenn er - trotz einiger dringender Wünsche - erst für dieses vierte Heft niedergelegt wurde, so spielte dabei die Oberlegung mit, daß jeder Leser bereits eine größere Zahl von Beispielen zur Hand haben sollte, wenn er die Ausführungen prüft. Es wäre also nicht angängig, ihn auf verstreut abgedruckte Quellen zu verweisen. Ganz sicher sind mit den Programmbeiträgen in den ersten vier Heften noch nicht alle empfehlenswerten Techniken erfaßt, es besteht aber ein ausreichender Fundus.

Unsere Darstellung betrifft Programmierung schlechthin, ihre Grundsätze beziehen sich eigentlich auf jeden Computer ebenso wie auf den Einsatz von Maschinen-Assembler, Interpretations- und Compilersprachen. Schwerpunktmäßig wird hier natürlich gleichwohl auf die 65xx-Mikroprozessoren und auf ihren Instruktionssatz abgestellt.

Der Systembenutzer bemerkt nach dem Studium der mitgelieferten Handbücher sehr schnell, daß sie ihm keinen unmittelbaren Lösungsweg für seine geplanten Anwendungen vermitteln. Das Anwenderhandbuch zeigt ihm, wie er das System in Betrieb nimmt und wie der das Betriebssystem, den Monitor, für den auch ein Programmlisting beigefügt ist, bedient. Das Hardware-Manual beschreibt Bausteine, Signale, Signalpegel, zeitliche Signalverläufe und Register in den Chips. Es handelt sich damit um wesentliche Information für den Ausbau und für das Interfacing der Hardware. Die Programmierfibel, das Programming Manual schließlich, beschreibt nur die Auswirkung von Befehlen und gibt einzelne beispielhafte kurze Instruktionsfolgen.

Der Anwender steht mit seiner Phantasie für das Programmieren allein da. Auch dieser Artikel wird an dieser Lage grundsätzlich nichts ändern können, es wird immer darauf ankommen, daß der Benutzer seine Zielvorstellungen selbst verwirklicht. Eine Zeitschrift wie diese oder ein Buch vermitteln Beispiele für häufig gebrauchte Problemlösungen und sie geben vielleicht Ordnungs- und Gestaltungsgesichtspunkte für die eigene Arbeit.

Die geschilderte 'Umgebung' ist für den Anwender bei den Mikros nicht anders als beim Einsatz eines IC, eines Taschenrechners oder eines Groß-

rechners: Er erhält ein funktionsfähiges Gerät, beschreibende Dokumentation mit Anwendungsvorschlägen und bei den Rechnern auch Systemprogramme (Monitore) und gegen Aufpreis ggfs. eine Bibliothek für Standardlösungen. Was er aus dieser Grundausstattung für seine Zwecke macht, das bleibt ihm allein überlassen.

Die Bedienung von Taschenrechnern ist dabei noch das einfachste Kapitel (mit zunehmendem Schwierigkeitsgrad!), weil sie dem dezimalen Rechnen dienen, das wir für viele Anwendungen in langen Schul- und Weiterbildungsjahren so gründlich gelernt haben. In die Bedienungsfolge, ob nun programmiert oder nicht, finden wir uns auf Grund von vorliegenden Formeln und bekannten Lösungswegen, den sog. Algorithmen, schnell ein.

> *Für größere Computer wurden Interpretations- und Compilersprachen sowie sehr leistungsfähige Betriebssysteme entwickelt, die den Anwender von einer Fülle notwendiger Dienstleistungsroutinen entlasten, die er für sein kleines Mikroprozessorsystem noch selber ausprogrammieren und einsetzen muß. Hier spielt sich bislang noch fast alles auf einer sehr einfachen Ebene ab.*

Programme können wir grob in folgende Gruppen einteilen:

a) Systemprogramme

Sie bieten allgemeine Dienstleistungen für die Systembedienung und für die Programmierung. Als Beispiele sind hier die auf den Entwicklungsplatinen enthaltenen Monitore zu nennen, die Datenein- und Ausgabe über Fernschreiber, Tonbandcassette bzw. über Tastatur und Leuchtanzeigen unterstützen. Dieser Gruppe sind viele der im 'First Book of KIM' und der in dieser Zeitschrift erschienenen Programme hinzuzurechnen, wie HYPERTAPE, BRANCH, BROWSE, MOVE-IT, RALOAD, QUICKDUMP und VERSALOAD, ZAHLENWANDLUNG, DISASSEMBLER usw. Schließlich auch Interpreter für BASIC und FOCAL, die verschiedenen Assembler, Text-Editoren, KIMMATH sowie Ein- und Ausgaberoutinen für spezielle Peripheriegeräte.

b) Demonstrations- und Spielprogramme

Die meisten Programme im 'First Book of KIM' gehören zu dieser Kategorie, ferner viele der vom Handel in Schrift- und Cassettenform beziehbaren Programme. Die meisten sind auf interaktive Bedienung abgestellt und erlauben ohne oder ohne umfangreiche Außenbeschaltung recht lustige Effekte an Leuchtanzeigen oder über die Lautsprecherausgabe (Musik).

c) Anwenderprogramme

Hier geht es um maßgeschneiderte Lösungen für besondere Anwendungen. In vielen Fällen wird die Hardware von der Standardausrüstung der CPU-Platine abweichen. Neue Ein- und Ausgabegeräte mit ihren Interfaces sind für die Zwecke digitaler Steuerung, Datenerfassung oder für kaufmännische Anwendungen zu bedienen. Man wird sich bemühen, die ausführenden Programme ohne unnötigen Ballast im System resident zu halten, wenn möglich in Festwertspeichern oder in schnell erreichbaren peripheren Speichern, wie Floppy Disk.

> *Anwenderprogramme sind so vielfältig wie das Leben selbst. Für sie gibt es keine allgemeinen Rezepte. Der Anwender muß den Lösungsweg selbst durchdenken und ihn für die Hardwarekonstellation selbst verwirklichen.*

Natürlich wird er sich dabei bemühen, eigene Teillösungen und ihm aus der Literatur bekanntgewordene Arbeiten und Beispiele von dritter Seite unter Abwandlung für seine Zwecke zu verwerten. Er wird sich weiter bemühen, sein Ziel ökonomisch hinsichtlich Speicher- und Zeitbedarf zu verwirklichen. Und er wird Vorgehens- und Programmierungstechniken verwenden, wie sie in diesem Artikel und in anderen beispielhaft vorgestellt worden sind.

Diese Arbeit bringt keine Wiederholung der Grundlagen, die ja ausreichend in den Systemhandbüchern und in zahlreichen Zeitschriftenaufsätzen und Büchern dargestellt worden sind. Sie erwartet vom Leser, daß er sich zumindest schon einmal selbst um die Schaffung eines eigenständigen kleinen Programmes bemüht hat, daß er die Auswirkung der Befehle kennt bzw. bereit ist, auf die entsprechenden Darstellungen im Verlaufe zurückzugreifen.

2. Das Ziel definieren

a) Die zeitliche Dimension

Ein Digitalrechner ist eine streng logisch aufgebaute Maschine, die in zeitlichen Verläufen arbeitet. Die Dimension 'Zeit' beinhaltet, daß es einen definierten Anfang und ein Ende geben muß. Dazwischen liegen Verläufe, die vom Programmierer gesteuert werden. Rein technisch dokumentiert sich das Element 'Zeit' durch das Vorhandensein eines Schwingkreises (meist mit Quarz), der den Maschinentakt, die Clocksignale, erzeugt.

In einem mechanischen System liegen die Funktionsabläufe in der Zeit naturgesetzlich fest; wenn man z.B. ein Pendel mit einem gewissen Impuls und unter einem bestimmten Winkel anstößt.

Diese Aussage kann man auch auf eine digitale elektronische Schaltung ausdehnen: Wenn man auf die Seite der Signaleingänge definierte Signale legt, so ist es eine Frage der Signalverknüpfung und der Gatterschaltzeiten, welche definierten Signale nach welcher Durchlaufzeit schließlich an den Ausgängen erscheinen. Auch dieses System ist 'deterministisch', es hat kein 'flexible response'. Digitalschaltungen erlauben das gleichzeitige Arbeiten mit beliebig vielen Bits an den Ein- und Ausgängen.

Nicht so der programmierbare Digitalrechner. Er ist frei hinsichtlich dessen, was er in der Zeit ausführen könnte. Begrenzungen bestehen eigentlich nur durch den Instruktionsvorrat und durch die Frequenz des Maschinentaktes. Sehr leger kann man auch etwa so definieren:

> *"Ein Computer ist eine aus sehr reinem Silizium hergestellte Maschine, die nach ihrem Einschalten fast nichts von sich aus tut, die wertvolle elektrische Energie verbraucht und die sich dabei noch nicht einmal etwas denkt."*

Und man muß hinzufügen: Die bisherigen 8-Bit-Mikros können pro Zyklus nur an 8 Bit arbeiten (Arbeitsbreite).

Zu einer deterministisch arbeitenden Maschine wird der Digitalrechner erst durch das Programm, daß seine Reaktionen unzweideutig festlegt und das ihm hinsichtlich der laufenden Anwendungen jeden Grad von Freiheit und Zufälligkeit abnimmt.

Die Maschine arbeitet also in der Zeit und in einer schaltungsmäßigen Umgebung. Welches elektrische Ausgangssignal bei gegebenem Eingangssignal erzeugt wird, hängt vom Willen des Programmierers ab. Auch der zeitliche Verlauf wird von ihm bestimmt. Die schnellstmögliche Reaktion hängt vom Maschinentakt ab, sie bewegt sich im Bereich der Mikrosekunden. Und die langsamstmögliche Reaktion findet in der Ewigkeit statt, wenn nicht zuvor Ausfälle eintreten sollten. Das Programm UNIVERSAL-TIMER (Heft 1, Seite 14) zeigt, daß man mit einer in nur 3 Byte niedergelegten Zählervorgabe bereits Zeitverzögerungen von 33 Jahren verwirklichen kann. Bei diesem Programm brauchte man nur die Vorgabekonstante zu ändern, um ein anderes Zeitverhalten zu bewirken. Im digitalen Schaltkreis müsste man stattdessen löten oder trimmen.

b) Die Sequenz: Eingabe - Verarbeitung - Ausgabe

Es wird nicht Ziel des Programmierers sein, den Rechner mit sich selbst zu beschäftigen, er soll stattdessen mit seiner Umwelt zusammenarbeiten. In der Prozeßsteuerung wird er also elektrische Eingangssignale zu steuernden Ausgangssignalen umsetzen sollen und ihre Pegel protokollieren und anzeigen, in Spielen wird er auf die Reaktion des Spielers an der Tastatur warten und seine vorprogrammierte Antwort zur Ausgabe bringen, in kaufmännischen Anwendungen wird er auf die Betätigung von Tastenfeldern durch den Bediener warten, von diesem aufgerufene Steuerfunktionen ausführen und schließlich Ergebnisse ausdrucken.

Wie auch beim digitalen Schaltkreis haben wir immer die Abfolge: Eingabe - Verarbeitung(Verknüpfung) - Ausgabe.

Ein- und Ausgabe müssen dabei nicht notwendig während des Programmablaufes erfolgen. Eingabedaten mögen zusammen mit dem Programm geladen worden sein, Ergebnisdaten werden wohlmöglich nicht an die Außenwelt abgegeben, sondern verbleiben im Speicher oder in Maschinenregistern. Dafür sei zum Beispiel auf die in Heft 2, S. 19 ff. abgedruckten Routinen zur ZAHLEN-WANDLUNG hingewiesen: Die Ergebnisse verbleiben im Speicher, wo sie für die Weiterverarbeitung, die Betrachtung mittels Monitor oder für die Ausgabe bereitstehen.

Ein Programm hat also zwischen Anfang und Ende einen zeitlichen Verlauf, der unter logischen Gesichtspunkten mit Konsequenzen für alle Eventualitäten festzulegen ist. Eingabedaten werden zu Ausgabedaten verarbeitet. - Für die Gestaltung des 'Dazwischen' gibt es nun bewährte Gesichtspunkte und Hilfsmittel.

c) Zieldefinition

Der Programmierer muß sich überlegen, was er innerhalb der zeitlichen Dimension und in welcher Reihenfolge erreichen will. In dieser Anfangsphase soll er sich nicht in Einzelheiten verlieren oder gar schon Programmteile in Maschinen-Assembler oder in einer höheren Sprache ausführen.

Er sollte in großen Blöcken denken und seine Gedanken in einfache Sätze kleiden und niederschreiben. Ein Beispiel aus Heft 2, S. 2: TRANSCRIBE HYPERTAPE TO QUICKDUMP "Programme oder Programmbibliotheken, die im Aufzeichnungsformat HYPERTAPE aufgezeichnet sind, sollen auf das doppelt so schnelle Aufzeichungsformat QUICKDUMP unter persönlicher Auswahl des Bedieners umkopiert werden können". Das ist eine allgemeine Zieldefinition.

Zweites Beispiel ASP (Heft 3, S. 13 ff.): "Es soll eine Reihe allgemein einsetzbarer Dienstleistungsroutinen für die arithmetische und logische Bearbeitung von Datenfeldern veränderlicher Länge angelegt werden".

Der Anfänger möge sich zu seiner eigenen Übung eine Reihe ihm zugänglicher Programme unter dem Gesichtspunkt ansehen, welche Zielsetzung der Autor verfolgt hat und diese Zielsetzung vielleicht auch mit eigenen Worten aufschreiben.

> *Die Analyse vorhandener Programme ist noch immer eine der ergiebigsten Lernquellen.*

d) Ein- und Ausgabedaten

> *Im zweiten Durchgang muß der Programmierer die Frage stellen, welche Eingabedaten ihm zur Verfügung stehen, ob es bei der Eingabe eine zeitliche Folge ihres Erscheinens aus vorhandenen Prozessbedingungen unveränderlich gibt oder zweckmäßigerweise geben sollte und schließlich, welche Ausgabe- oder Steuerdaten in zeitlicher Reihenfolge daraus gebildet werden müssen.*

Nehmen wir als anschauliches Beispiel die kaufmännische Rechnungsschreibung. Hier gibt es häufig die Abfolge: Eingabe der Kundennummer mit gleichzeitigem Ausdruck - Ausdruck von Kundennamen und Adresse aus dem Speicher - E/A der Artikelnummer - Ausdruck der Artikelbezeichnung aus dem Speicher - E/A der Menge - Verarbeitung auch des Rabattsatzes - Abdruck der Ergebnisse. Nach der letzten Artikelzeile werden Mehrwertsteuer und Gesamtbetrag gerechnet und abgedruckt, die Summen werden in einen Tageszähler übernommen.

Die Aufeinanderfolge von Ein- und Ausgabe und die dazwischenliegenden Verarbeitungsschritte sind durch die kaufmännische 'Verarbeitungslogik' vorgezeichnet. Besonderheiten ergeben sich aus dem Aufbau der Dateien, aus der Formularanordnung und aus den Verarbeitungswünschen.

Für den Programmierer bedingt diese Logik, daß von der Tastatur herkommende Eingabedaten in den Stadien des 'Prozeßfortschrittes' eine unterschiedliche Bedeutung haben und damit unterschiedlich interpretiert werden müssen. Sie führen zu verschiedenen Ausgaben auf dem Drucker oder in den Speicher.

e) Analyse der Bedingungen im zeitlichen Verlauf

Das Beispiel der Rechungsschreibung zeigt für die blockmäßige Analyse durch den Programmierer auch Bedingungen an. Ich kann den Endbetrag für die Artikelzeile erst dann berechnen, wenn zuvor folgende Daten zur Verfügung gestanden haben: Kundennummer (sie führt auch auf den Rabattsatz des Kunden), Artikelnummer (Preis) und Menge.

Oder das Beispiel TRANSSCRIBE HYPERTAPE TO QUICKDUMP: Erst ein vollständig geladenes Programm (Eingabe) soll zur Ausgabe von Startadresse und ID auf den Leuchtanzeigen des Systems führen. Dieses soll auf einen Tastendruck (Eingabe) warten. Ist er Null, so soll das System den nächsten Bandsatz laden, ist er ungleich Null, so soll mit der Routine QUICKDUMP aus dem Speicher auf Band geschrieben werden (Ausgabe).

> *Der Programmierer tut gut daran, die Aufgabestellung als eine Verknüpfung von Bedingungen im Zeitablauf zu sehen und sie mit einfachen Sätzen zu formulieren: 'Was muß zuerst geschehen sein, ehe eine Verarbeitung einsetzen kann ?'*

Die zeitliche Abfolge Eingabe - Verarbeitung - Ausgabe wird fast immer den Vorrang vor der rechen- und maschinenlogischen Ausführung im einzelnen haben.

> *Die Analyse der zeitlichen Ablaufbedingungen in einfachen Satzaussagen führt zu einer fast automatischen Gliederung der Problemlösung in überschaubare Funktionsblöcke. Eine gute Vorbereitung in dieser Phase erspart spätere Umstellungen bei der Programmniederschrift in ihren Einzelheiten.*

Man kann es auch so formulieren: Solange die geforderten Bedingungen im Zeitablauf noch nicht erreicht sind, muß die Maschine in eine Warteschleife geführt werden, die das Eintreten der Bedingung abfragt. Diese Aussage gilt für Bedingungen, die von der Außenwelt auf die Maschine zukommen, also z.B. für die Tastaturbetätigung oder das 'handshake' einer Peripherieeinheit. In dieser Wartezeit kann man die Kapazität der Maschine auch für andere Überwachungs- und Verarbeitungsaufgaben einsetzen, man muß nur in einer sich dann ergebenden größeren Schleife mit ausreichender Häufigkeit das Eintreten der speziellen Bedingung überprüfen.

Etwas anders sieht es aus, wenn das Eintreten der Bedingung auf Interrupt geschaltet ist. Das elektrische Interruptsignal veranlaßt die Maschine, die gerade ausgeführte Programmfolge zu verlassen und an den Beginn einer Instruktionsfolge zu verzweigen, die man mit Interruptroutine bezeichnet.

Und auch folgendes ist zu verdeutlichen: Eben fragten wir, ob Bedingungen in der Außenwelt eingetreten seien. Aber bestimmt ebenso oft muß die Maschine prüfen, ob sie durch ihre Tätigkeit in der Zeit selbst die Bedingung geschaffen hat, um einen eingeleiteten Ausführungsblock verlassen zu können, um zur nächsten Tätigkeit übergehen zu dürfen. Nehmen wir den Transport von 8 Bytes von einem Speicherbereich zu einem anderen. Ein wie immer gearteter Zähler wird anfangs auf 8 gesetzt, nach jedem Transport wird er um 1 vermindert, und es wird geprüft, ob der Zähler schon auf Null ist. Wenn nicht, muß die Maschine weitere Transporte durchführen, ehe sie zu den Folgeinstruktionen weiterschreiten kann. Hier heißt die Bedingung also nicht 'warte auf das Eintreten einer äußeren Bedingung', sondern 'fahre in der Tätigkeit fort, bis die Bedingung erreicht ist'. Ähnlich sind die Vorgaben bei Rechenoperationen. Z.B. 'Berechne den Sinus des Arguments X nach der Formel solange, bis in der achten Ergebnisstelle keine Veränderung mehr eintritt'.

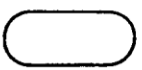
Jede der vorgenannten Anweisungen läßt sich in einem Satz unserer Sprache ausdrücken, der einen kleinen Funktionsblock beschreibt. Z.B.:

'Lade einen Satz vom Magnetband von Anfang bis Ende'.
'Setze einen Speicherbereich auf definierte Anfangswerte'.
'Warte auf das Eintreten einer äußeren Bedingung'.
'Arbeite an einer Programmfolge solange, bis eine Bedingung erreicht ist'.

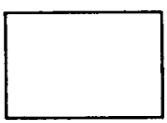
> *Die Gliederung einer umfangreicheren Aufgabe in eine Reihe so einfacher Satzaussagen als Tätigkeitsbeschreibungen führt zu einem Programmablaufplan. Er stellt eine allgemeine Übersicht für auszuführende Prozeduren dar.*

f) Programmablaufpläne

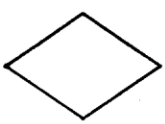
Eine graphische Darstellung der Tätigkeitsbeschreibungen ist sehr zu empfehlen. Für sie gibt es Zeichenschablonen und sogar eine Norm. Die wichtigsten graphischen Darstellungselemente sind:



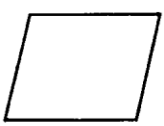
GRENZSTELLE, Z.B. ANFANG ODER ENDE
AUCH ZWISCHENHALT



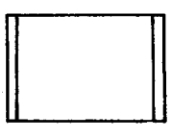
OPERATION, ALLGEMEIN, INSBES. FÜR OPERATIONEN, DIE IM FOLGENDEN NICHT BESONDERS AUFGEFÜHRT SIND



PROGRAMMVERZWEIGUNG NACH ABFRAGE
IM SONDERFALL: VERZWEIGUNG DURCH PROGRAMMIERTEN SCHALTER

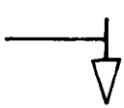


EINGABE ODER AUSGABE, MASCHINELL ODER VON HAND



BENUTZUNG EINES UNTERPROGRAMMES

ÜBERGANGS-, ANSCHLUSS-STELLE



ABLAUFLINIE, ZUSAMMENFÜHRUNG UND RICHTUNGSPFEIL

Programmablaufpläne mit diesen graphischen Elementen finden wir an verschiedenen Stellen im Programmier-Handbuch und auch im Anwender-Handbuch.

Der Leser möge bemerken, daß man diese Programmablaufpläne anfangs für große Tätigkeitsblöcke entwirft und dann zu immer weiteren Verfeinerungen fortschreitet ggfs. bis hin zu einzelnen kritischen Maschineninstruktionen, wie z.B. an einigen Stellen im Programmier-Handbuch.

3. Einfluß der Hardwarekonstellation auf die Programmierung

Soweit ein Programm auf einem vorhandenen Computersystem ohne zusätzliche Außenbeschaltung laufen soll, z.B. auf einem Entwicklungssystem wie KIM-1, SYM-1, AIM 65 etc, liegen die verfügbaren Speicherbereiche, die Adressen der Peripherie für Ein- und Ausgabe und die Adressen des residenten Monitorprogrammes fest. Anzusprechende Adressen sind aus den Handbüchern bekannt, und es werden im allgemeinen für den Zweck passende Routinen zur Bedienung der Peripherie im Monitor enthalten sein.

Die erste Besonderheit tritt auf, sobald man einen der vorhandenen Timer im Interruptbetrieb einsetzen will. In Heft 3, S. 22 ff. wurde auf die Besonderheiten dieser Betriebsart hingewiesen.

> *Interruptbeschaltung bedingt notwendig, daß es ein Vordergrund- und ein Hintergrund- (Interrupt-) Programm geben muß. Die Problemlösung ist damit in mindestens zwei Ebenen anzugehen.*

Man wird sich überlegen, für welche Phasen des Vordergrundprogrammes man die Interruptmöglichkeit durch den Befehl SEI aufhebt, damit keine unerwünschten Störungen auftreten. Innerhalb der Interruptroutine wird man im allgemeinen anfangs die Inhalte der Register A, X und Y auf den Stack retten und am Ende wieder herstellen.

Die nächsten Besonderheiten treten beim Anschluß weiterer Geräte und Schaltkreise an die bekannte vorhandene Platine auf. Sobald an die Ports von Peripheriebausteinen wie z.B. 6520 OIA, 6532 VIA etc. Ein- oder Ausgabeleitungen gelegt werden,

> *müssen bei Programmbeginn definierte Anfangsbedingungen hinsichtlich der Funktion der Ports als Ein- oder Ausgang geschaffen werden.*

Man erreicht dies durch Schreiben in das Datenrichtungsregister. Gleiche Vorkehrungen betreffen die Kontrollregister, mit denen die Funktion der Kontroll-Leitungen festgelegt wird. Und zugleich tritt wieder die Frage auf, ob man mit der erweiterten Peripherie im Interruptbetrieb fahren will oder nicht (s.o.).

Sofern man die Peripherie in messenden Anordnungen vom Hauptprogramm aus bedienen will (also ohne Interrupt), so wird man die Peripheriebausteine regelmäßig abzufragen haben, ob sich das Bitmuster an den Ports oder die Flags in den Kontrollregistern verändert haben.

> *Zusammenarbeit mit der Peripherie bedeutet immer, daß man einen Übersetzer für den vom Gerät gesendeten oder von ihm erwarteten Code (parallel, seriell, Geschwindigkeit) bereithalten muß.*

Im KIM-Monitor z.B. kennen wir solche Programme für die Bedienung der Tastatur, für die Multiplexansteuerung der Leuchtanzeigen, für das Schreiben und Lesen auf Magnetband und für den Verkehr mit TTY. Jede dieser E/A-Einheiten arbeitet mit einer spezifischen Geschwindigkeit und mit einem anderen Code als dem rein binären der Zentraleinheit. Schon wieder andere Verhältnisse finden wir an den Leuchtanzeigen des AIM 65. Es handelt sich da um die 'intelligente'DL1416A, die ASCII-Code direkt dekodiert und die einzelnen Anzeigen selbsttägig multiplext. Wieder andere Verhältnisse finden bei einem ASCII-Keyboard, bei Druckern, Bildschirm etc. In der Konsequenz können wir etwa wie folgt zusammenfassen:

Der Anschluß zusätzlicher Schaltkreise und Peripheriegeräte fordert vom Programmierer Initialisierungsroutinen, Codeumsetzungsprogramme zur Interpretation und zur Steuerung und ggfs. auch die Zuweisung besonderer Speicherbereiche für die Ein- und Ausgabe. Interruptbetrieb bedeutet Programmierung auf mehreren Ebenen.

Diese Aussage bleibt bei der Erweiterung des Computers durch Peripheriebausteine und Speicher unverändert gültig; es ist dann jedoch die Zuweisung der Speicheradressenbereiche nach den Empfehlungen des Herstellers sorgfältig zu planen.

Der Anwender wird sich im Vorwege überlegen, welche Peripherie und welche Speicher er für die Erfüllung seiner Aufgabe braucht. Die Art des Anschlusses der Leitungen an Ports und Kontrolleingänge beeinflußt die Programmierung aber nicht nur hinsichtlich des Interruptbetriebes sondern auch hinsichtlich ihres Umfanges und ihrer Eleganz. Gesetzt den Fall, zwei Eingangs-Signalleitungen seien auf die Ports PAØ und PA1 gelegt. Dann würde man zur Feststellung, ob z.B. PAØ gesetzt ist, wie folgt schreiben:

```
LDA #$01   DER FRÜHERE AKKU-INHALT WIRD ZERSTÖRT DURCH LADEN
BIT PAD    ABFRAGE DES PORTS UNTER LOGISCH AND
BNE ...    VERZWEIGE, WENN BIT GESETZT IST
```

Eine gleiche Sequenz müßte man abfahren, um festzustellen, ob speziell PA1 gesetzt ist. Hinzu kämen ggfs. Maßnahmen zur Rettung und Wiederherstellung des Akku-Inhaltes. - Legt man dagegen die Eingangsleitungen auf die hohen Bits PA6 und PA7, so kann man ohne Zerstörung des Akkuinhaltes kurz abfragen:

```
BIT PAD    INSTRUKTION SETZT GGFS. FLAGS FÜR NEGATIVE, OVERFLOW, ZERO
BMI ...    VERZWEIGE, WENN PA7 GESETZT
BVS ...    VERZWEIGE, WENN PA6 GESETZT
```

Dieses kleine Beispiel zeigt, wie sehr Hardware und Programmierung voneinander abhängig sind. Oder man will beispielsweise auf bis zu 256 Leitungen ausgeben. Soll man dann 16 Bausteine 6520 mit je 2x8 Ports einsetzen? Sicher nicht! Wenn die zeitlichen Verhältnisse es erlauben, wird man 2 Stück billige 4 zu 16 Dekoder an nur einen 8-Bit-Port anschließen und jeden dieser Dekoder mit einem Halbbyte ansteuern. Man braucht allerdings dann auch eine weitere äußere Beschaltung an den Kreuzungspunkten (logisch UND).

Die Verdrahtung der Peripherie beeinflußt die Möglichkeiten zu sparsamer Programmierung.

4. Einfluß der beabsichtigten Programmverwendung

Es macht einen großen Unterschied für die Anlage eines Programmes, ob es auf der Platine eines Entwicklungssystemes mit seinem Monitor oder ob es in einer anderen Umgebung laufen soll. Im ersten Fall führt das RESET beim Einschalten der Betriebsspannung auf den Monitor mit seinen Möglichkeiten des Programmladens und des Einstellens auf die Startadresse des Programmes. Dem Anwenderprogramm selbst stehen nützliche Unterroutinen aus dem Monitor, z.B. für die Codewandlung zur Verfügung. Nach seinem Ablauf kann ein Anwenderprogramm die Kontrolle an den Monitor zurückgeben.

Bei einer für feststehende Anwendungen konzipierten Computerplatine muß das RESET nach dem Einschalten (man kann für automatisches RESET ein Monoflop vorsehen) auf den Anfang des ausführenden Programmes hinweisen. Der Vektor dieses Startpunktes muß durch eine Hardwareschaltung oder durch einen Festwertspeicher unter der Adresse zu finden sein, die dieses System mit FFFC und FFFD dekodiert.

Feststehende Anwendung wird auch bedeuten, daß das Programm in einem Festwertspeicher abgelegt ist (ROM, PROM, EROM). Für den Programmaufbau heißt dieses weiterhin, daß alle Arbeitsspeicher im Adressenbereich des RAMs liegen müssen, sie können sich also nicht in einer noch nicht voll genutzten page des Programmbereiches befinden. - Festwertspeicher bedeutet weiterhin: Programm-Modifikationen durch Einsetzen neuer Opcodes oder Adressen in vorhandene Befehle sind nicht möglich. (Ausnahme: Man transportiert einen Programmabschnitt anfangs ins RAM und modifiziert dort. Sprünge sind dann erforderlich.)

Ein Programmbetrieb in einer solchen vom Entwicklungssystem losgelösten Umgebung macht es erforderlich, alle etwa benötigten und sonst im Monitor nutzbaren Hilfsroutinen in den neuen Festwertspeicher einzucodieren.

> *Soll ein Programm neben anderen auf einer Maschine laufen, so muß es in seiner Speicherverwendung auf die anderen Programme Rücksicht nehmen, um nicht deren Werte unbeabsichtigt zu zerstören. Das gilt auch in Hinsicht auf die Speicherverwendung des Monitorprogrammes, speziell in der Zeropage.*
>
> *Programme, die auf allen 65xx-Systemen laufen sollen, dürfen nicht feststehende und ganz bestimmte Funktionen ausübende Adressen eines einzelnen Systemes heranziehen.*

5. Der zweckmäßige Aufbau eines Programmes

Nach der Programmablaufplanung, den Oberlegungen zur Beschaltung der Hardware und ihren Einfluß auf die Programmierung, den Erwägungen, in welcher Umgebung das Programm laufen soll, können wir uns nun eingehender mit dem Programmaufbau befassen. Programmiert und codiert wird auch an dieser Stelle noch nicht; es werden allgemein bewährte Techniken dargestellt.

Dem Anfänger sei hier gesagt, daß er jede Freiheit zur Unordnung und zu einem persönlichen Wildweststil im Programmaufbau hat, wenn er nur auf

seine Weise konsequent bleibt und logisch vorgeht. Der CPU ist das einerlei. Er wird bei dieser Vorgehensweise bereits nach zwei Wochen sein eigenes Programm nicht ohne längere Interpretation nachvollziehen können. Und kaum ein anderer Betreiber dieser Systeme wird ihm auf seinen einsamen Höhenflug zu folgen vermögen und vor Bewunderung rufen 'quelle élégance'.

Ein Programm sollte etwa folgende Struktur haben:

INITIALISIERUNGSPHASE

Ein- und Ausgabeleitungen werden durch Schreiben in die Kontroll- und Richtungsregister in die beabsichtigte Funktionsweise versetzt.

Speicherbereiche, Pointer und Softwareschalter werden auf die benötigten Anfangswerte gebracht.

Interruptvektoren werden geladen.

AUSFÜHRUNGSPHASE

a) Hauptprogramm, das den Kern des zeitlich-logischen Flusses enthält.

b) Interruptprogramm(e)

c) Sammlung der Unterprogramme für a) und b). Sie leisten wiederkehrende Dienste, besonders auch für die Codewandlung an der Ein- und Ausgabe.

KONSTANTEN- UND TABELLENBEREICH

ARBEITSSPEICHERBEREICH

Die Anordnung ist übersichtlich, weil die Instruktionen im Zusammenhang bleiben (Initialisierungs- und Ausführungsphase) und weil darauf ein unveränderlicher Teil mit Konstanten und Tabellen folgt, der zum Programm gehört und mit ihm zusammen geladen werden muß bzw. mit ihm zusammen in einem Festwertspeicher enthalten ist.

Der Arbeitsspeicher ist auf jeden Fall ein unabhängiges Gebilde. Er muß nicht auf das Programm folgen, seine Benutzung kann mit anderen Programmen geteilt werden, wie es ja sehr häufig in der Zeropage geschieht. Bei zeitkritischen Anwendungen wird er überwiegend in der Zeropage liegen. Im übrigen sei auch auf Heft 2, Seite 14 hingewiesen.

An den Schluß des Hauptprogrammes werden sicher Maßnahmen gehören, die die in der Initialisierungsphase eingeschalteten Peripheriegeräte wieder ausschalten und ihnen insbesondere die Interruptmöglichkeiten nehmen. Ferner wird vom Benutzer ein Signal erwartet, daß das Programm beendet ist. Das kann auch durch die Übergabe der Kontrolle an den Monitor geschehen.

Ein sorgfältig bearbeitetes Programm wird sich neben diesem Aufbau nicht nur durch seine zuverlässigen Dienste für alle logisch denkbaren Eventualitäten auszeichnen, sondern auch durch die lückenlose Besetzung der Speicherstellen mit Instruktionen - und erst daran anschließend mit Konstanten und Tabellen.

Es wäre kein guter Programmierstil, Teilabschnitte des Programms jeweils an einer mit Null endenden Adresse beginnen zu lassen und die sich bildenden Lücken mit 'no operation' oder mit Sprüngen zu überbrücken. Man würde dabei Speicherraum verschenken. Den Programmierer selbst mag das nicht stören, das Programm verliert aber an Wert für die Nutzung durch andere Verwender, die auf Speicherökonomie oder Durchführungszeiten achten müssen.

Es wäre fast schon kriminell, mitten in die Instruktionsfolge Festwerte (Konstanten) oder Arbeitsspeicher zu legen.

Für die Beachtung dieser Disziplin gibt es folgende wichtige Gründe: Der Programmaufbau unterläge von Mal zu Mal Zufälligkeiten, die das Wiederlesen erschweren. Ferner: Man muß stets damit rechnen, daß ein Programm zu einem späteren Zeitpunkt zu verändern ist. Ein schlechter Aufbau wird zu weiteren Behelfslösungen führen und er birgt die Gefahr in sich, daß zu ändernde Felder übersehen werden.

Die Fülle der bereits vorhandenen Dienstprogramme für Programmierung und Verschieblichkeit setzt den oben empfohlenen Aufbau voraus (z.B. RELOCATE und MINI-DIS im 'First Book of KIM', sowie RALOAD, VERSALOAD u.a. in dieser Zeitschrift). Insbesondere bemerkt es ein Disassembler nicht, wenn er in eine Tabelle hineinstolpert. Er wird die vorgefundenen binären Muster weiterhin als Instruktionen ausdeuten. Für das Auslisten eines Programmes ist das ausgesprochen hinderlich.

Der Leser möge sich durch die Analyse der ihm zugänglichen Programme davon überzeugen, daß es sich hier um ein überall praktiziertes und bewährtes Aufbauprinzip handelt.

Sicher wird er hier und da Verstöße von der Regel finden, die möglicherweise aus der Beschränkung des verfügbaren Speicherraumes herrühren. Man halte sich z.B. für den KIM-1 vor Augen, daß er 4 RAM-Adreßbereiche hat, die voneinander getrennt sind, also nicht durchgängig zur Verfügung stehen. Page 0 wird mit dem Monitor geteilt, der hintere Teil von page 1 dient dem Stack, page 2 und 3 sind unbeschränkt verfügbar, in page 17 ist dann noch ein kleines Fenster frei. Wenn man ein langes Programm in einer solchen Umgebung ansiedeln muß, dann sollte man seine Anordnung besonders behutsam planen.

Ein Hinweis noch zum Initialisierungsprogramm: Das Schreiben in die Kontrollregister etc. und das Laden der Interruptvektoren könnte man auch manuell mit Hilfe der Monitoreingabe besorgen. Dieser Weg ist nicht empfehlenswert, denn manuelle Eingaben sind fehlerbehaftet. Man besorge daher das entsprechende Laden vom Programm aus.

6. Die Niederschrift eines Programmes

Die bisherigen Ausführungen betrafen Rahmenplanungen vor der eigentlichen Niederschrift: Zielsetzung, Programmablaufplanung, Einflüsse der Beschaltung und Struktur des Programmes. Wenn damit auch eine Ordnung in das Vorgehen gebracht wurde, so ist damit des Planens noch kein Ende.

> *Der Programmierer sollte sich in dieser Phase von den Grundsätzen der Einmaligkeit und der Zeit- und Speicherökonomie leiten lassen.*

a) Speicherzuweisung

Bei der vorausgeschätzten Länge des zu schreibenden Programmes ist festzulegen, wo es seinen Speicherplatz finden soll, ohne daß es andere auch systemresidente Programme stört. Bei zeitkritischen Programmen wird man ihren Arbeitsspeicher soweit als nötig und möglich in die Zeropage legen. Das sich dieser Zeropage bedienende Programm muß selbst natürlich nicht dort angesiedelt sein, es kann an beliebiger Stelle im RAM oder in Festwertspeichern stehen.

Pointer für die indirekte Adressierung mit X oder Y müssen auf jeden Fall in der Zeropage liegen. Der besondere Befehl STX, adressiert durch den Inhalt von Y, funktioniert nur in der Zeropage. Sollen seine eleganten Möglichkeiten ausgenutzt werden, so ist dort also Platz für die Abspeicherung vorzusehen.

Speicherzellen, die in einem Programm häufig angesprochen werden, sollte man in der Zeropage ansiedeln, denn man spart pro Instruktion 1 Byte gegenüber absoluter Adressierung, vom Zeitfaktor einmal ganz abgesehen. Solche Zellen sind Zähler, logische Schalter, Register für den Vergleich, Rechenfelder, Abspeicherungszellen für die Maschinenregister nach JSR oder nach Interrupt usw.

> *Man achte darauf, daß man in der Zeropage nicht andere Programme, auch nicht den Monitor stört.*

Page 1 steht nur beschränkt zur Verfügung, abhängig von der Verschachtelungstiefe von Unterprogrammen und Interrupten, weiterhin abhängig von der Nutzung des Stacks für Parameterübergabe. Eine Zerstörung von Speicherinhalten kann hier durch Fehlbedienung der Maschine eintreten.

b) Das Prinzip der Einmaligkeit

Beim Programmentwurf wird von Anfang an klar geworden sein, daß man eine Reihe häufig wiederkehrender Abläufe als Unterprogramm anlegen sollte (Speicherökonomie). Fast regelmäßig gehören die Ein- und Ausgaberoutinen sowie die Codewandlung dazu.

Bei der Niederschrift des Hauptprogrammes und auch bei der Bearbeitung von Interruptroutinen und sogar von Unterprogrammen wird man gleichwohl wiederholt feststellen, daß man gleiche oder fast die gleiche Instruktionsfolge schon einmal codiert hat. In diesem Falle wird man seine bisherige Planung revidieren und die betreffende Instruktionsfolge als ein weiteres Unterprogramm anlegen. Dieses sollte man dabei vorausschauend möglichst vielseitig einsetzbar machen, um auch ähnliche künftige Fälle abzudecken. Oft genug kann dieses Ziel durch die Wahl des Einsprungspunktes in das Unterprogramm erreicht werden. Verwiesen sei hier z.B. auf das Unterprogramm FIND in ROLDIS (Heft 3, S. 36).

Die gleiche Philosophie sollte man auch auf Unterprogramme anwenden. Es ist dem Programmierer unverwehrt, Unterprogramme ineinander zu verschachteln oder in eine Zählschleife einzusetzen (siehe z.B. RAM TEST WITH RANDOM PATTERNS, Heft 3, Seite 26, Programmabschnitte FCHECK und RCHECK). An diesem Beispiel möge der Leser auch ersehen, wie wunderbar kurz und klar gegliedert ein steuerndes Hauptprogramm bei Beachtung der Grundsätze sein kann.

c) Die Speicherökonomie

Die Zweckmäßigkeit von Programmschleifen (loops) dürfte dem Leser aus fast jedem veröffentlichten Programm und aus den Handbüchern bekannt sein. Aber auch solche Schleifen legt man nur einmal in ihrer ganz spezifischen Form für eine bestimmte Dienstleistung an, die in der Art ihrer Zählung oder Adressierung im Programm einmalig ist. Wenn in einem Programm häufiger gleichartige Dienstleistungen gebraucht werden, sollte man sich um die Zusammenfassung in einem gemeinsamen Unterprogramm bemühen.

Beispiel: Im einen Fall sollen 5 Bytes und in einem zweiten Fall 8 Bytes usw. von einer Stelle 'VON' aus zu einer Stelle 'NACH' abwärts transportiert, addiert, verglichen usw. werden. In diesem Falle hängt die Zahl der Bytes von der Stelle im Programm ab. Die Dienstleistung als solche ist immer gleich. Also übergibt man das Längenattribut in einem der Indexregister X oder Y und ruft ein gemeinsames Unterprogramm 'DO' auf. Z.B.

```
LDX #$ LÄNGE
JSR DO.
```

Innerhalb des Unterprogrammes wird man X dabei als Addresser und Zähler einsetzen. Häufiger aber noch wird man Dienstleistungen benötigen, in denen man den Zähl- und den Adressiervorgang (indizierte Adressierung) voneinander unabhängig halten möchte. In solchen Fällen fällt es nicht schwer, das eine Indexregister für den Zählvorgang und das andere für die Adressierung einzusetzen:

```
LDX #$ LÄNGE
LDY #$ ADDRESSER
JSR DO
...
```

Bei einem Transportvorgang (MOVE) hätte das Unterprogramm dann z.B. folgendes Aussehen:

```
DO    LDA VON,Y      Bei Pointeradressierung:DO LDA (VON),Y
      STA NACH,Y                               STA (NACH),Y
      DEY                                      DEY
      DEX                                      DEX
      BNE DO                                   BNE DO
      RTS                                      RTS
```

Man wird natürlich die Häufigkeit der Ansprache des Unterprogrammes prüfen, um festzustellen, ob insgesamt eine Verminderung der benötigten Bytes möglich ist (s.a. Heft 3, Seite 2). Und man wird weiterhin prüfen, ob hier Y als Addresser alle in Frage kommenden Sende- und Empfangsfelder überstreichen kann, ggfs. wird man sie entsprechend anordnen.

Zweckmäßigkeitsuntersuchungen der vorgenannten Art, auch in Bezug darauf, welches Indexregister als Zähler und welches als Addresser zu wählen ist oder ob man etwa eine Zelle in der Zeropage als Zähler nehmen sollte, wird man häufig genug erst bei der Programmniderschrift anstellen können. Dann aber sollte man sich für gründliche Lösungen entscheiden.

Nur bei sehr zeitkritischen Anwendungen wird man von diesen Grundsätzen der Einmaligkeit und der Speicherökonomie abweichen und bestimmte Instruktionsfolgen in einem Programm öfter als einmal linear ausprogrammieren. Es sei vermerkt, daß der Mikroprozessor in den meisten Anwendungen so wenig ausgelastet ist, daß er auf auslösende Ereignisse 'lange' warten muß.

Die Grundsätze der Einmaligkeit und der Speicherökonomie führen zu zwei weiteren Oberlegungen: Wenn im residenten Monitor oder in anderen residenten Programmen, auch einer Library, bereits Teilprogramme enthalten sind, die das neue Programm unterstützen oder entlasten können, so sollte man sie einsetzen. Und umgekehrt:

> *Man sollte nützliche Dienstleistungen grundsätzlich so schreiben, daß sie auch von anderen residenten Programmen mitbenutzt, in eine Programmbibliothek eingestellt oder in spätere Programme als Baustein eingefügt werden können.*

Eine Lösung sollte immer möglichst universal sein. Der Programmierer entlastet sich durch die Zahl der gründlich erledigten Teilprobleme.

Und schließlich sei auf die zahlreichen Dienstleistungen im Advanced Subroutine Package und in den Makros hingewiesen (Hefte 2-4 dieses Journals). Warum also nicht solche Module einsetzen und sie mit VERSALOAD (Heft 2) ausführungsfähig zusammenschweißen (linken)?

d) Die eigentliche Niederschrift

Nun kommen wir zu der Stelle, an der der Programmierer so ziemlich allein vor seinem Problem steht. Zu seiner Entlastung werden wir ihn in den nachfolgenden Abschnitten noch auf zahlreiche bewährte Programmiertechniken und auf häufig gemachte Fehler hinweisen.

Nach den systematisierenden und ordnenden Vorbereitungen sollte nun eigentlich nichts mehr schief gehen, gleichwohl sind auch versierte Programmierer nicht immer aufmerksam genug und müssen sich per 'trial and error' durchkämpfen. Es ist wohl etwas eine Frage der Erfahrung und des Zutrauens, wie man vorgeht. Man mag oben im Programmablaufplan beginnen, das Hauptprogramm zuerst ausfüllen und hernach die Subroutines. Oft genug mag man mit einer zentralen und sehr schwierigen Routine anfangen, für die man noch keine Vorbilder hat. Oder man beginnt bei den kleinsten Bausteinen, den Unterprogrammen.

Jede Art des Vorgehens legt Anforderungen an die anderen Programmteile und für die Art der Weitergabe von Werten und Zwischenergebnissen fest. Wenn ein Unterprogramm seine Ergebnisse z.B. in den Registern A, X und Y abliefert, so muß das Hauptprogramm mit diesen Registerinhalten weiterarbeiten - und umgekehrt: Wenn das Hauptprogramm als zuerst fertiggestellter Teil die Ergebnisübergabe in Registern oder in einem Speicherfeld fordert, dann muß das Unterprogramm sie dorthin abliefern.

Die 65xx-Prozessoren haben 2 Indexregister (derzeitige 8-Bit-Mikros). Oft genug reicht das für Zähl- und Adressierungszwecke nicht aus. Dann kann man auch Speicherzellen für Zählzwecke einsetzen, indem man inkrementiert, dekrementiert oder Werte arithmetisch addiert oder subtrahiert. Letztere Programmierung ist aber weniger elegant.

Manche Unterprogramme werden beide Indexregister verwenden müssen. Als Beispiel betrachte man die KIM-Monitorroutine SCAND für die Anzeige auf den LEDs. Sie beginnt in Programmzeile 1057 bei Speicherzelle 1F19. Ihr Rumpf endet mit Zeile 1080. Die Hauptroutine ruft zweimal das Unterpro-

gramm CONVD ab Zeile 1085 auf. Wir sehen, daß beide Programmteile mit X arbeiten. Die Hauptroutine setzt den Anfangswert für X, das Unterprogramm erhöht jeweils. Beide arbeiten geplant an einem Strang. Beim Y-Register ist es anders. In Zeile 1064 wird Y als Zähler mit 03 geladen, in Zeile 1087 wird aber ein ganz anderer im Akku aufbereiteter Wert für die Adressierung mit Index Y benötigt. Also muß zuvor in Zeile 1085 Y vorübergehend abgespeichert und in Zeile 1099 wiederhergestellt werden. Mit einem dritten Indexregister hätte man eleganter arbeiten können.

> *Die noch laufende Verwendung eines Registers in einer Schleife des aufrufenden Programmes und der Einsatz desselben Registers für andere Zwecke in einem Unterprogramm bedingen Maßnahmen zur Sicherung und späteren Wiederherstellung des Registerinhaltes.*

Der zweckmäßige Gebrauch der Indexregister in einem Unterprogramm ist schon einige Überlegungen wert. Man kann allgemein sagen, daß es für die Indizierung mit X etwa doppelt soviele Instruktionen wie für die Indizierung mit Y gibt, insbesondere auch die Schiebe- und Rollbefehle und das Zeropage,X. Gleichwohl erlaubt Y die so elegante Adressierung über einen Pointer in der Zeropage. Diese Adressierungsart stellt wohl den größten Fortschritt gegenüber den 6800-CPUs dar.

Man wird das Wechselspiel der Indexregister also sorgfältig planen. Es wird damit auch klar, daß manches für die zeitlich frühere Anlage der ausführenden Unterprogramme spricht, um ihnen den jeweils wirkungsvollsten Instruktionssatz zur Verfügung stellen zu können. In diesem Fall müsste sich das Hauptprogramm am bestehenden Engpaß orientieren. Zum Beispiel sei auf die Funktion des Y-Registers in den Makros hingewiesen. In Heft 3, S. 8 an der Programmstelle GEN dient Y als Addresser und Zähler. Zunächst lädt es absolut indiziert das X-Register als weiteren Addresser, dann lädt Y den Akku über Pointer-Adressierung und schließlich wird Y um eins heruntergezählt. Eine so diffizile kurze Operation kann man mit X nicht bewirken, also widmet man dem Einsatz von Y besondere Aufmerksamkeit. Eine ähnliche Planung findet man im ASP, Hefte 3 u. 4, verwirklicht.

> *Der Programmierer tut gut daran, jeweils überschaubare kleine Programmabschnitte in Arbeit zu nehmen und sie nach ihrer Niederschrift zu testen.*

Es ist fast ein Naturgesetz, daß ein Programm noch während seiner Entwicklung verändert werden muß. Das Naturgesetzliche geht dabei vom menschlichen Wesen aus, das aus seinem Lernprozeß neue Zweckmäßigkeitserwägungen ableitet, von Flüchtigkeiten einmal ganz abgesehen.

> *Insofern sollte man dem Programmierer empfehlen, mit seinem Programm überhaupt erst einmal anzufangen und Fehler zu machen, auch wenn ihm alle Instruktionsschritte noch nicht ganz klar vor Augen stehen.*

Wer seine Maschine recht gut beherrscht, hat fast immer zunächst Versuche in langen Stunden am Gerät gemacht. Es ist hier nicht anders als beim Autofahren, beim Aufbau einer Schaltung oder beim Klavierspielen.

Man schreibe auf, was man programmiert, und zwar mit Kommentaren. Zu schnell sind sonst die Oberlegungen vergessen. Vorbilder mögen die Programm-Listings dieser Zeitschrift sein. Und man sollte sich zur eigenen Ordnung ein Programmierformular schaffen, das etwa wie folgt aufgebaut ist:

QUELLENPROGRAMM					PAGE	INSTRUKTION		
MARKE	OPcode	Mode	Operand	Bemerkungen	AD	OP	low	high

Man fängt mit dem Eintragen der mnemonischen Opcodes und der geplanten Adressierungsart an, ferner mit den Bemerkungen. Anfangs-, Verzweigungs- und Einsprungspunkte sowie Operanden erhalten frei erfundene aber möglichst sinngebende (symbolische) Namen. Die Möglichkeit, Namen zu verwenden, erleichtert die Arbeit wesentlich. Sie macht u.a. den Komfort höherer Programmiersprachen aus. Wir sollten daher in ausreichendem Umfange Namen verwenden, auch wenn wir von Hand assemblieren, um die Lesbarkeit und Sicherheit zu verbessern.

Sofern wir einen der erhältlichen Assembler verwenden, die aus mnemonischen Instruktionen den Maschinencode erzeugen, so sind wir sogar gezwungen, die entsprechenden Stellen und Operanden mit Namen zu versehen.

> *Bei der Niederschrift der Instruktionen sollten wir uns zunächst noch nicht um den Maschinen-Opcode und um die Adressen im Befehl und auch noch nicht um die Adresse kümmern, unter der der Befehl abgespeichert werden soll.*

Denn es ist ziemlich wahrscheinlich, daß die Reihenfolge der Instruktionen aus Gründen der Zweckmäßigkeit noch etwas abgeändert wird, daß Instruktionen fortfallen oder hinzukommen. Bei jeder dieser Änderungen müßten wir dann auch viele Abspeicherungsadressen und Adressen in den Instruktionen abändern. Die damit verbundene Schreib- und Rechenarbeit kann lästig werden und zu Fehlern führen. Das gleiche gilt für Sprungweiten in den Branches und für die Sprung- und Unterprogrammadressen. Man lasse diese Kästchen (Bytes) auf dem Formular zunächst leer, kennzeichne sie vielleicht farbig, um sie in Erinnerung zu halten.

Erst wenn wir mit dem mnemonischen Code soweit zufrieden sind, daß wir eine erfolgreiche Programmausführung erwarten dürfen, sollten wir die Maschineninstruktionen hinzufügen, dabei die geplante Adressierungsart genau beachtend. Die Länge der Maschinenbefehle führt automatisch auf die von ihnen besetzten Speicherplätze.

Fortsetzung folgt mit vielen Beispielen zur Programmiertechnik. *R.L.*

'HIDDEN OPCODE 9C'

Im Programm SUMMARY, Heft 2, S. 17, 2. Instruktionszeile, wurde der 'hidden opcode' 9C verwandt, um ein mit X indiziertes Datenfeld auf Null zu löschen. Dieser Befehl arbeitet nur dann einwandfrei, wenn das Y-Register definiert auf 00 steht. Eine Einsparung von Code läßt sich also nicht erzielen.

Herr Werner Wöhr, Kellersteige 17 in 7078 Aalen-Uko, hat dankenswerterweise auf diesen Umstand aufmerksam gemacht. Nach seinen hier inzwischen bestätigten Feststellungen findet bei Y ungleich Null offensichtlich ein logisches AND zwischen Y und dem dritten Byte der Instruktion statt, das beim AND um eins erhöht verwertet wird. Das Ergebnis dieser logischen Operation wird dann wie vorgesehen mit X indiziert abgespeichert. Eine praktische Verwertbarkeit des Opcodes 9C für den normalen Gebrauch ist daher nicht abzusehen.

****

THE 65XX FAMILY: DIE NEUEN PROZESSOREN

Die Auslieferung der ersten Systeme AIM 65 wird leider erst im Januar 1979 beginnen. Über die Preise (plus MWSt) liegen hier folgende Informationen vor: DM 875,- mit 1 kRAM, DM 1.050,- mit 4 kRAM, Assembler im ROM DM 230,-, BASIC-Interpreter im ROM DM 280,-. Ob die BASIC-ROMs schon von Anfang an geliefert werden, ist nicht ganz sicher. Auf der ELECTRONICA wurde auch ein Selbst-Test-Programm für den AIM 65 im Festwertspeicher vorgeführt. Konditionen sind noch nicht bekannt.

Hinsichtlich der Stromversorgung des AIM 65 haben sich die Anforderungen nach den neuesten Prospekten vereinfacht: +5 V ± 5% bei 2,0 A (max.), 24 V ± 15% unreguliert bei 2,5 A Spitze, 0,5 A Durchschnitt. 12 Volt sind also nicht mehr erforderlich. - Dem Entwurf für das Anwenderhandbuch war zu entnehmen, daß 1,0 A gebraucht werden, wenn es sich um die 1 kRAM-Platine handelt, die mit 2 ROMs Betriebssystem bestückt ist und wenn das Display dunkel ist. Mehr als 1,5 A werden gezogen von der 4 kRAM-Platine mit 5 ROMs und voll erleuchtetem Display.

Die Systemhandbücher werden zunächst nur in englischer Sprache vorliegen.

Synertek's Einkarten-Computer heißt jetzt SYM-1, nachdem es in den USA Einspruch wegen des vorgesehenen Namens VIM-1 gegeben hatte. Die Geräte werden bereits ausgeliefert und kosten DM 789,- (+MWSt). Statische RAM-Karten zu seiner Erweiterung werden von Electronic 2000 wie folgt angeboten: 4k DM 478,-, 8k DM 699,-, 12k DM 889,- und 16k DM 1.081,-. Angekündigt ist ein Bausatz mit 2 Floppy-Disk-Laufwerken (Shugart S 400) zu DM 2.998,-.

****

Der Vertrieb des AIM 65 wurde inzwischen auch von folgenden Firmen übernommen: GWK Technische Elektronik, Aachener Str. 56, 5132 Übach-Palenberg, Tel.02451-41911 - MICRO-SHOP BODENSEE, Postfach 11 22, 7778 Markdorf, Tel.07544-3575 - FELTRON Elektronik Vertrieb, Postfach 11 69, 5110 Troisdorf-Spich, Tel. 02241-41004.

****

Literaturhinweis: Die hier oft zitierten Zeitschriften KILOBAUD und MICRO, beide USA, können über den Fachzeitschriftenvertrieb M. Nedela bezogen werden. Anschrift: Marktstraße 3, 7778 Markdorf 1, Tel. 07544-3575.

A S P - Advanced Subroutine Package (Teil 3)

Michael Zimmermann, Eberstädter Str. 170, 6102 Pfungstadt

E: The 'extended subroutines' of this third part operate with two length attributes, one for each operator. They implement decimal multiplication and division, signed, and include services prior and after execution of the main routines.

Inhaltsübersicht für den 3. Teil

* * *

2. Erweiterte Unterprogramme

Im Gegensatz zu den einfachen Unterprogrammen, die nur mit Operanden gleicher Länge arbeiten, ist bei den erweiterten Unterprogrammen auch die Verarbeitung von Operanden unterschiedlicher Länge möglich.

Alle Operationen werden in dezimaler Form durchgeführt, eine Beschränkung, die angesichts der Einsatzschwerpunkte von ASP nicht so gravierend sein dürfte, da eine binäre Arithmetik sich auf Adreßrechnungen beschränken dürfte. Alle Operationen werden vorzeichengerecht durchgeführt.

Die erweiterten Unterprogramme verwenden eine Reihe von internen Routinen, deren Aufrufadressen den Programmauflistungen entnommen werden sollten und die den spezifischen Benutzerbedürfnissen angepaßt werden müssen.

Die erweiterten Unterprogramme enthalten im 1-byte-langen Längenfeld des Aufrufes in den ersten vier Bitstellen die Länge des ersten Operanden, in den folgenden vier Bitstellen die Länge des zweiten Operanden.

2.1 Internes Unterprogramm der Längenberechnung (LEN)

Alle erweiterten Unterprogramme sind in der Lage, Operanden mit unterschiedlicher Länge zu bearbeiten, der Längenschlüssel im 1. Byte des Aufrufes muß also aufbereitet werden. Hierzu dient das nachfolgende Unterprogramm, welches den Längenschlüssel nach der Parameterübernehme aus E2 lädt, aufsplittet und die Länge des ersten Operanden in EA und die des zweiten in E9 ablegt.

```
XX00  A5 E2     LEN     LDA E2          LOAD LEN FIELD
  02  29 0F             AND #$0F        MASK OFF FIRST BITS
  04  85 E9             STA E9          STORE LEN FOR 1RST OPERAND
  06  A5 E2             LDA E2          LOAD LEN FIELD AGAIN
```

```
XX08  4A 4A          LSR,LSR
  0A  4A 4A          LSR,LSR
  0C  85 EA          STA EA          STORE AT LEN 2ND OPERAND
  0E  60             RTS
```

2.2 Internes Unterprogramm zum Löschen des Rechenspeichers (CLR)

Für die erweiterten Unterprogramme ist es erforderlich, den Teil der Page 0, der von diesen Routinen angesprochen wird, vor Inanspruchnahme zu löschen. Diese Routine setzt daher die Speicheradressen 00C0 bis E1 auf den Wert 00.

```
XX00  A9 00      CLR   LDA #$ØØ       LOAD ZERO
  02  A2 20            LDX #$20       SET COUNT
  04  95 C0      C10   STA C0,X       STORE VALUE IN WORKAREA
  06  CA               DEX            DECREMENT COUNTER
  07  10 FB            BPL C10        NOT DONE, GOBACK
  09  60               RTS
```

2.3 Internes Unterprogramm der Datenübernahme (MVOR)

Für die Anwendung der erweiterten Unterprogramme müssen die Daten im Zwischenbereich stehen. Zugleich auch ist für Multiplikation und Division die Normierung der Vorzeichen erforderlich.

In diesem Unterprogramm wird das Vorzeichen des Datenfeldes festgestellt und in Abhängigkeit vom Ergebnis entweder eine direkte Obertragung oder eine Subtraktion von Null vorgenommen.

Die Adresse des Datenfeldes, aus dem übertragen werden soll, steht in E7, der Bereich, in den die Werte abgelegt werden sollen, steht im X-Register, die Länge in Y und das Vorzeichen im Akkumulator. Die entsprechenden Register, bzw. der Adreßpointer müssen durch die aufrufende ASP-Routine gefüllt werden.

Bei einem negativen Vorzeichen des Datenfeldes wird E1 um 1 erhöht, stellt sich also auf E1=1, wenn nur eines der beiden Felder negativ ist. E1=0 oder =2, wenn keines oder wenn beide Datenfelder negativ sind. Eine Analyse von E1 findet im Unterprogramm zur Datenabgabe statt und dient dazu, das Vorzeichen des Ergebnisses zu steuern.

```
XX00  C9 99      MVOR  CMP #$99       CHECK SIGN OF FIELD
  02  F0 0A            BEQ LMIN       MINUS, GOTO LMIN
  04  B1 E7      M10   LDA (E7),Y     LOAD DATA FIELD
  06  95 00            STA 00,X       STORE IN WORKAREA
  08  CA               DEX            DECREMENT COUNTER
  09  88               DEY            AND POINTER ADDRESSING
  0A  10 F8            BPL M10        NOT DONE, NEXT BYTE
  0C  30 0E            BMI MEND       DONE, GOTO END

  0E  38               SEC            SET CARRY
  0F  F8               SED            SET DECIMAL
  10  A9 00      L20   LDA #$ØØ       SET COMPLEMENT VALUE
  12  F1 E7            SBC (E7),Y     SUBTRACT DATA FIELD
  14  95 00            STA 00,X       STORE IN WORKAREA
  16  CA               DEX            COUNTER -1
  17  88               DEY            ADDRESSING -1
```

```
XX18  10 F5           BPL      NEXT BYTE IF NOT DONE
  1A  E6 E1           INC E1   SET NEGATIVE FLAG
  1C  60        MEND  RTS      RETURN
```

2.4 Internes Unterprogramm zur Datenabgabe (MNACH)

Nachdem im Verlaufe eines erweiterten Unterprogrammes im Pufferbereich eine Operation durchgeführt wurde, sind die Daten aus dem Zwischenspeicher in das Ergebnisfeld abzulegen. Ebenso ist bei Multiplikation und Division das Vorzeichen des Ergebnisses zu berücksichtigen.

Das Vorzeichen wird zuerst durch Prüfen der Speicherstelle E1 bestimmt und entsprechend verzweigt. Bei positivem Vorzeichen erfolgt eine reine Übertragung, bei negativem eine Subtraktion von Null.

Die Länge der Übertragung bzw. der Subtraktion muß in EB stehen und durch das aufrufende Programm entsprechend gefüllt werden.

Die höchste Speicherzelle, aus der das Ergebnis entnommen wird, steht im X-Register, das Ziel der Übertragung in E3. Diese Datenzellen sind vor der internen Verarbeitung durch das aufrufende ASP-Modul entsprechend zu füllen.

```
XX00  A4 EB     MNACH  LDY EB          LOAD LEN OF RESULT
  02  A5 E1            LDA E1          LOAD NEGATIVE FLAG
  04  29 01            AND #$01        AND TEST
  06  D0 0A            BNE NMIN        IF NEGATIVE GOTO NMIN

  08  B5 00      N10   LDA 00,X        LOAD RESULT FROM WORKA
  0A  91 E3            STA (E3),Y      STORE IN RESULT
  0C  CA 88            DEX,DEY         COUNTER & POINTER-ADDR. -1
  0E  10 F8            BPL N10         NOT DONE, NEXT BYTE
  10  30 0C            BMI NEND        DONE, GOTO END

  12  38        NMIN   SEC             SET CARRY
  13  F8               SED             SET DECIMAL
  14  A9 00      N20   LDA #$00        SET COMPLEMENT VALUE
  16  F5 00            SBC 00,X        SUBTRACT RESULT
  18  91 E3            STA (E3),Y      & STORE
  1A  CA 88            DEX,DEY         -1 AS ABOVE
  1C  10 F6            BPL N20         NOT DONE, NEXT BYTE
  1E  60        NEND   RTS             DONE, RETURN
```

2.5 Dezimale Multiplikation (MULT)

Mit diesem Unterprogramm können zwei dezimale Zahlen mit einer Länge von jeweils bis zu 8 Bytes multipliziert werden. Die Länge des Ergebnisfeldes wird durch die Addition der beiden Faktoren-Feldlängen ermittelt. Dementsprechend ist für den ersten Operanden, der auch das Ergebnis aufnimmt, ein entsprechend großes Feld vorzusehen.

Zuerst werden die Parameter in den Zwischenbereich übernommen, der Rechenbereich gelöscht, die Längen der Operanden separiert und die Länge des Ergebnisses berechnet und in EB abgestellt.

Danach werden die Parameter für die Datenübernahme in die entsprechenden Felder geladen und dann übernommen, der erste Operand in D8-DF, der zweite Operand in DØ bis D7.

Die eigentliche Multiplikation wird sodann im Rechenfeld durchgeführt. Hierzu wird der erste Operand solange in das Ergebnisfeld C8-CF addiert, bis die erste Stelle des zweiten Operanden Null ist. Ist diese Bedingung erreicht, so werden der zweite Operand und das Ergebnis um 1 Stelle nach links verschoben.

Dieser Durchlauf erfolgt insgesamt achtmal, entsprechend der maximalen Länge eines der Operanden. Aus dem Ergebnisfeld wird zum Abschluß der Wert wieder in den Operanden 1 übertragen.

```
XX00 20 XX XX    MULT    JSR MVOR        TAKE PARAMETERS
  03 20 XX XX            JSR CLR         CLEAR WORKAREA
  06 20 XX XX            JSR LEN         COMPUTE LEN OF OPS
  09 D8                  CLD             COMPUTE LEN OF RESULT
  0A 38                  SEC
  0B A5 E9               LDA E9
  0D 65 EA               ADC EA
  0F 85 EB               STA EB
  11 A5 E3               LDA E3          SET PARAMETERS
  13 85 E7               STA E7          FOR TAKING FIRST OP TO
  15 A5 E4               LDA E4          WORKAREA
  17 85 E8               STA E8
  19 A2 00               LDX #$00
  1B A1 E7               LDA (E7,X)
  1D A4 E9               LDY E9
  1F A2 D7               LDX D7
  21 20 XX XX            JSR MVOR
  24 A5 E5               LDA E5          SET PARAMETERS
  26 85 E7               STA E7          FOR TAKING 2ND OP TO
  28 A5 E6               LDA E6          WORKAREA
  2A 85 E8               STA E8
  2C A2 00               LDX #$00
  2E A1 E7               LDA (E7,X)
  30 A4 EA               LDY EA
  32 A2 DF               LDX #$DF
  34 20 XX XX            JSR MVOR
  37 A0 07         MLT   LDY #$07        SET COUNT
  39 A5 D0         MØ5   LDA D0          CHECK FIRST BYTE OF OP
  3B F0 21               BEQ M20         IF ZERO GOTO M20
  3D F8                  SED
  3E 18                  CLC             ADD SECOND OP TO RESULT
  3F A2 07               LDX #$07
  41 B5 C8         M10   LDA C8,X
  43 75 D8               ADC D8,X
  45 95 C8               STA C8,X
  47 CA                  DEX
  48 10 F7               BPL M10
  4A A2 07               LDX #$07        CARRY RESULT INTO HI ORDER
  4C B5 C0         M15   LDA C0,X        POSITIONS
  4E 69 00               ADC #$00
  50 95 C0               STA C0,X
  52 CA                  DEX
  53 10 F7               BPL M15
  55 38                  SEC             DECREMENT FIRST BYTE
  56 A5 D0               LDA D0          FIRST OP
  58 E9 01               SBC #$01
```

```
XX5A  85 D0          STA D0
  5C  D0 DB          BNE MØ5        IF NOT ZERO ADD ONCE MORE
  5E  A2 00    M20   LDX #$00
  60  B5 C1    M25   LDA C1,X       SHIFT RESULT & FIRST OP
  62  95 C0          STA C0,X       1 BYTE TO LEFT
  64  E8             INX
  65  E0 18          CPX #$18
  67  D0 F7          BNE M25
  69  A9 00          LDA #$00
  6B  85 CF          STA CF
  6D  88       M30   DEY            ALL SHIFTS DONE ?
  6E  10 C9          BPL MØ5        NO ?
  70  A2 CF          LDX #$CF       SET PARAMETERS AND
  72  20 XX XX       JSR MNACH      TRANSFER RESULT
  75  60             RTS
```

2.6 Dezimale Division

Mit diesem Unterprogramm können zwei dezimale Zahlen mit einer Länge von je bis zu 8 Bytes dividiert werden.

Die Länge des Ergebnisfeldes wird durch Subtraktion der beiden Feldlängenattribute gewonnen, der Quotient steht im Dividendenfeld linksbündig.

Zuerst werden die Parameter in den Zwischenbereich übernommen, der Rechenbereich gelöscht, die Operandenlängen bestimmt, ebenso die Länge des Quotienten, die nach EB abgestellt wird. Danach erfolgt Obernahme der Operanden in den Rechenbereich, zuerst der Dividend nach C7 bis CE, dann der Divisor nach D7 bis DF.

Anschließend erfolgt Prüfung des Divisors auf Null. Tritt dieser Wert auf, verbotene Division, so erfolgt Rückkehr in das aufrufende Programm mit FF im Akkumulator.

Mit den geprüften positiven Operanden wird dann die eigentliche Division vollzogen. Hierbei wird jedesmal der Divisor vom Dividenden abgezogen. Ergibt sich hierbei für den Dividenden ein Wert kleiner als Null, so wird eine Korrekturaddition durchgeführt. Hernach erfolgt Linksverschiebung von Dividend als auch von Quotient um 1 Byte nach links. Es wird weiter subtrahiert, bis die Gesamtzahl der Bytes erreicht ist. - Ergibt sich bei der Subtraktion ein positiver Dividend, so wird der Quotient um 1 erhöht und es wird mit der nächsten Subtraktion fortgefahren.

Nach Vollzug von 8 Zyklen dieser Art wird das Ergebnis (Quotient) linksbündig in den Dividenden eingestellt; ein möglicher Rest steht in CØ bis C7.

```
XX00  20 XX XX   DVD   JSR MVOR       TAKE PARAMETERS
  03  20 XX XX         JSR CLR        CLEAR WORKA
  06  20 XX XX         JSR LEN        COMPUTE LEN OF OPS
  09  D8               CLD            COMPUTE LEN OF RESULT
  0A  38           SEC  SEC
  0B  A5 EA            LDA EA
  0D  E5 E9            SBC E9
  0F  85 EB            STA EB
  11  A5 E3            LDA E3         DIVIDEND TO WORKA
  13  85 E7            STA E7
  15  A5 E4            LDA E4
  17  85 E8            STA E8
```

```
XX19  A2 00          LDX #$00
  1B  A1 E7          LDA (E7,X)
  1D  A4 EA          LDY EA
  1E  A2 CE          LDX #$CE
  21  20 XX XX       JSR MVOR
  24  A5 E5          LDA E5          TAKE DIVISOR TO WORKA
  26  85 E7          STA E7
  28  A5 E6          LDA E6
  2A  85 E8          STA E8
  2C  A2 00          LDX #$00
  2E  A1 E7          LDA (E7,X)
  30  A4 E9          LDY E9
  32  A2 DF          LDX #$DF
  34  20 XX XX       JSR MVOR

  37  A2 07          LDX #$07        CHECK DIVISOR: ZERO?
  39  B5 D8    DV1Ø  LDA D8,X
  3B  D0 06          BNE DIV
  3D  CA             DEX
  3E  10 F9          BPL DV10
  40  A9 FF          LDA #$FF        FLAG FOR ZERO DIVISOR
  42  60             RTS             EXIT TO CALLER

  44  A0 07    DIV   LDY #$07
  46  F8       DØ5   SED             SUBTRACT DIVISOR FROM
  47  38             SEC             DIVIDEND
  48  A2 07          LDX #$07
  4A  B5 C0    D10   LDA C0,X
  4C  F5 D8          SBC D8,X
  4E  95 C0          STA C0,X
  50  CA             DEX
  51  10 F7          BPL D10
  53  90 0E          BCC D20         IF BELOW ZERO GOTO CORRECTIVE ADD
  55  38             SEC             ADD 1 TO RESULT
  56  A2 07          LDX #$07
  58  B5 D0    D15   LDA D0,X
  5A  69 00          ADC #$00
  5C  95 D0          STA D0,X
  5E  CA             DEX
  5F  10 F7          BPL D15
  61  30 E3          BMI DØ5         SUBTRACT ONCE MORE

  63  18       D20   CLC             DO CORRECTIVE ADD
  64  A2 07          LDX #$07        TO DIVIDEND
  66  B5 C0    D25   LDA C0,X
  68  75 D8          ADC D8,X
  6A  95 C0          STA C0,X
  6C  CA             DEX
  6D  10 F7          BPL D25
  6F  A2 00          LDX #$00

  71  B5 C1    D30   LDA C1,X        SHIFT DIVIDEND & QUOTIENT
  73  95 C0          STA C0,X        1 BYTE TO LEFT
  75  E8             INX
  76  E0 18          CPX #$18
  78  D0 F7          BNE D30
```

Berichtigung zum ASP in Heft 3, Seite 17: Im Abschnitt 1.5, Mitte der Seite muß es richtig heißen

BEQ wenn Operand 1 gleich Operand 2

```
XX7A  A9 00        LDA #$00      CLEAR RIGHTMOST BYTE
  7C  85 D7        STA D7
  7E  88           DEY           DECREMENT COUNTER
  7F  10 C5        BPL DØ5       NOT DONE, DO IS  AGAIN
  81  A2 07        LDX #$07
  83  20 XX XX     JSR MNACH
  86  60           RTS
```

[WIRD FORTGESETZT]

****

PRINTERPROGRAMM FÜR MINI-DOT

Ingo Dohmann, Im Wiehagen 15, 6830 Gütersloh 12

E: The alphanumeric (64 characters) Mini-Dot-Printer 7706B which is marketed by the Logitec GmbH, Kaiser-Wilhelm-Strasse 26 in D-8130 Starnberg, is driven by this set of subroutines. As an application for this 7x5 matrix printer a hex dump routine HEXDOT is supplied.

Beim Logitec Mini-Dot-Printer handelt es sich um einen Metallpapierdrucker mit 7x5 Punktmatrix und einem Zeichenvorrat von 64 alphanumerischen Zeichen (Modell 77068). Er ist für den direkten Anschluß an ein PIA ausgelegt. Für den Anschluß an KIM-1 in diesen Unterprogrammen sind außer der Stromversorgung von +5 Volt, 0,2 A und -24 Volt, 0,2 A folgende Verbindungen herzustellen:

```
Port A Bit 0-5 an Drucker Pin 8 - 13
"    A Bit 6   "  "       Pin H
"    A Bit 7   "  "       Pin 5
```

Die Druckgeschwindigkeit beträgt 2 Zeilen, entspr. 64 Zeichen pro Sek.

Die nachfolgenden utility-Unterprogramme bringen folgende Dienste:

```
Adresse 1200 Print Space
        1204 Print CR/LF
        1206 Print ASCII Character
        1251 Zeilenpuffer löschen (Zellen 0100-011F)
        1265 Print Byte
```

Um wahlweise auch über TTY ausgeben zu können, übernimmt die Zelle 17FC eine Steuerfunktion: 17FC=00 Druckerausgabe, 17FC=FF TTY-Ausgabe
In der Zeropage werden die Zellen FD und EB zur Rettung der Indexregister verwandt.

```
1200  A9 20       DOT1   LDA #$20      FÜR DRUCK ZWISCHENRAUM
1202  DO 02              BNE 1206      VERZWEIGE IMMER
1204  A9 0A              LDA #$ 0A     LADE FÜR CR/LF WAGENRÜCKLAUF/
1206  86 FD              STX FD        REGISTER SICHERN ¤ ZEILENTRANSP.
1208  84 EB              STY EB

120A  2C FC 17           BIT 17FC      DRUCKER ODER TTY ?
120D  F0 0A              BEQ 1219      DRUCKER!
120F  C9 0A              CMP #$ 0A     FÜR TTY
1211  F0 03              BEQ 1219
1213  4C A0 1E           JMP OUTCH     KIM-ROUTINE
```

```
1216  4C 2F 1E          JMP CRLF       KIM-ROUTINE
1219  C9 0A             CMP #$0A       WENN LF, DANN DRUCKEN
121B  F0 15             BEQ 1232
121D  C9 0D             CMP #$0D       CR WIRD IGNORIERT
121F  F0 0C             BEQ 122D
1221  AE 20 01          LDX 0120
1224  9D 00 01          STA 0100,X     ABLAGE IM PUFFER
1227  CA                DEX
1228  30 08             BMI DOTZEI     DRUCKEN, WENN PUFFER VOLL
122A  8E 20 01          STX 0120       AKTUELLEN STAND SPEICHERN
122D  A6 FD             LDX FD         REGISTER ZURÜCKHOLEN
122F  A4 EB             LDY EB
1231  60                RTS

1232  A9 BF       DOTZEI LDA #$BF      UPRO: DRUCKE ZEILE
1234  8D 01 17           STA PADD       DATENRICHTUNGSREGISTER
1237  A2 1F              LDX #$1F       PUFFERZÄHLER SETZEN
1239  BD 00 01           LDA 0100,X     HOLE ZEICHEN VOM PUFFER
123C  8D 00 17           STA PAD        SCHREIBE ZUM PRINTER
123F  2C 00 17           BIT PAD        WARTEN AUF NÄCHSTE ANFORDERUNG
1242  10 FB              BPL 123F
1244  2C 00 17           BIT PAD
1247  30 FB              BMI 1244
1249  CA                 DEX
124A  10 ED              BPL 1239       PUFFER AUSGESCHRIEBEN?
124C  A9 A0              LDA #$A0       DRUCKER AUSSCHALTEN
124E  8D 00 17           STA PAD

1251  A9 20              LDA #$20       'ZWISCHENRAUM'
1253  A2 1F              LDX #$1F       ALS ZÄHLER
1255  9D 00 01           STA 0100,X     LÖSCHE DEN PUFFER
1258  CA                 DEX
1259  10 FA              BPL 1255
125B  A9 1F              LDA #$1F
125D  8D 20 01           STA 0120       ZEICHENZÄHLER NEU GESETZT
1260  A6 FD              LDX FD         REGISTER ZURÜCK
1262  A4 EB              LDY EB
1264  60                 RTS

1265  2C FC 17    DOTBYT BIT 17FC       PRINTER ODER TTY?
1268  F0 03              BEQ 126D       DRUCKER!
126A  4C 3B 1E           JMP PRTBYT     KIM-ROUTINE FÜR TTY
126D  85 FC              STA FC         AKKU RETTEN
126F  4A 4A              LSR,LSR
1271  4A 4A              LSR,LSR        ISOLIERE VORDERE 4 BITS
1273  20 78 12           JSR 1278
1276  A5 FC              LDA FC         AKKU ZURÜCK
1278  29 0F              AND #$0F       ISOLIERE RECHTE 4 BITS
127A  C9 0A              CMP #$0A       UMWANDLUNG IN ASCII
127C  18                 CLC
127D  30 02              BMI 1281
127F  69 07              ADC #$07
1281  69 30              ADC #$30
1283  4C 06 12    END    JMP 1206
```

Literaturhinweis: 'The Best of MICRO, Volume 1', eine Zusammenfassung der nicht mehr lieferbaren Hefte 1-6 von MICRO, ca. 160 Seiten stark, ist jetzt im Fachzeitschriftenvertrieb Nedela, Marktstr. 3, 7778 Markdorf zu etwa DM 30,- als Buch erhältlich.

H E X D O T

Dieses ebenfalls von Herrn Dohmann geschriebene Programm ergibt über den Mini-Dot-Printer einen hexadezimalen Speicherauszug (dump).

```
12EC  D8          HEXDOT  CLD
12ED  AD F5 17            LDA SAL         TRANSPORT DER VORGABEN AUS
12F0  85 80               STA $80         SAL/SAH ZUR POINTERADRESSE
12F2  AD F6 17            LDA SAH
12F5  85 81               STA $81
12F7  38                  SEC
12F8  AD F7 17            LDA EAL         ANFANG UND ENDE SINNVOLL GESETZT?
12FB  ED F5 17            SBC SAL
12FE  AD F8 17            LDA EAH
1301  ED F6 17            SBC SAH
1304  B0 11               BCS $1317       AUSDRUCK NUR WENN DIFFERENZ POSITIV
1306  XX                                  ANDERNFALLS AUSDRUCK EINES HIN-
                                          WEISTEXTES AUS EINER TABELLE, HIER
                                          NICHT AUSGEFÜHRT.
1317  20 51 12            JSR $1251       LÖSCHE AUSGABEPUFFER
131A  20 64 13            JSR $1364       DRUCKE POINTER
131D  A0 00               LDY #$00
131F  A5 80               LDA $80         LEERZEICHEN BIS 1. STELLE
1321  29 07               AND #$07
1323  F0 17               BEQ $133C
1325  85 82               STA $82
1327  0A                  ASL             ANZAHL DER LEERZEICHEN ×3
1328  18                  CLC
1329  65 82               ADC $82
132B  AA          TAX     TAX
132C  20 00 12            JSR $1200       AUSGABE LEERZEICHEN
132F  CA                  DEX
1330  D0 FA               BNE $132C       DURCH?
1332  F0 06               BEQ $133A
1334  20 04 12            JSR $1204
1337  20 64 13            JSR $1364       POINTER-AUSGABE
133A  A0 00               LDY #$00
133C  B1 80               LDA ($80),Y     LADE ZEICHEN
133E  20 65 12            JSR $1265       DRUCKE ALS ASCII (2 ZEICHEN)
1341  20 00 12            JSR $1200       AUSGABE LEERZEICHEN
1344  E6 80               INC $0080       ERHÖHE POINTER
1346  D0 02               BNE $134A       SKIP ON NO CARRY
1348  E6 81               INC $0081
134A  A5 80               LDA $80         FERTIG?
134C  CD F7 17            CMP EAL
134F  D0 07               BNE $1358       SKIP
1351  A5 81               LDA $81         POINTER HI
1353  CD F8 17            CMP EAH
1356  F0 08               BEQ $1360       WENN ENDVORGABE ERREICHT
1358  A5 80               LDA $80         ZEILE VOLL?
135A  29 07               AND #$07
135C  D0 DC               BNE $133A       WEITERE ZEICHENAUSGABE IN DER ZEILE
135E  F0 D4               BEQ $1334       NEUE ZEILE
1360  20 04 12            JSR $1204       ZEILENTRANSPORT
1363  00                  BRK             ENDE DES AUSDRUCKES
```

```
1364  A5 81          LDA $81        UNTERPROGRAMM FÜR DIE AUSGABE
1366  20 65 12       JSR $1265      DES LAUFENDEN POINTERSTANDES
1369  A5 80          LDA $80        POINTER LOW
136B  20 65 12       JSR DOTBYT     DRUCKE BYTE
136E  20 00 12       JSR $1200      DRUCKE ZWISCHENRAUM
1371  20 00 12       JSR $1200      DITO
1374  60             RTS
```

T Y D U M P

Vom Inhalt der Zelle 17FC hängt bei diesen Programmen von Herrn Dohmann ab, ob über Mini-Dot-Printer oder über Teletype ausgegeben wird. Für einen Speicherauszug über TTY folgt hier das Pendant zu HEXDOT.

```
128A  2C FC 17   TYDUMP  BIT $17FC       DRUCKER ODER TTY?
128D  D0 03              BNE $1292       SKIP FÜR TTY
128F  4C EC 12           JMP HEXDOT      FÜR DRUCKERAUSGABE

1292  78                 SEI
1293  D8                 CLD
1294  AD F5 17           LDA SAL         POINTER INITIALISIEREN
1297  85 80              STA $0080
1299  AD F6 17           LDA SAH
129C  85 81              STA $81
129E  A9 10              LDA #$10        ZÄHLER FÜR 1 SEITE
12A0  85 82              STA $82
12A2  20 2F 1E           JSR CRLF        KIM-ROUTINE F. TTY-AUSGABE
12A5  A2 00              LDX #$00
12A7  A5 81              LDA $81
12A9  20 3B 1E           JSR PRTBYT      AUSGABE VON 2 ASCII FÜR 1 HEX
12AC  A5 80              LDA $80
12AE  20 3B 1E           JSR PRTBYT
12B1  20 9E 1E           JSR OUTSP       ZWISCHENRAUM
12B4  20 9E 1E           JSR OUTSP
12B7  A5 80              LDA $80
12B9  29 0F              AND #$0F        MASKIEREN
12BB  85 83              STA $83         LEERZEICHEN BIS ZUM 1. BYTE
12BD  E4 83              CPX $83
12BF  F0 07              BEQ $12C8
12C1  20 9E 1E           JSR OUTSP
12C4  E8                 INX
12C5  18                 CLC
12C6  90 E9              BCC $12B1
12C8  A2 00              LDX #$00
12CA  A1 80              LDA ($80,X)     1 BYTE AUSGEBEN
12CC  20 3B 1E           JSR PRTBYT
12CF  20 9E 1E           JSR OUTSP       ZWISCHENRAUM
12D2  E6 80              INC $80         POINTER ERHÖHEN
12D4  D0 02              BNE $12D8
12D6  E6 81              INC $81
12D8  A5 80              LDA $80
12DA  29 0F              AND #$0F        MASKIEREN
12DC  D0 EA              BNE $12C8
12DE  C6 82              DEC $82
12E0  D0 C0              BNE $12A2
12E2  20 5A 1E           JSR GETCH
12E5  C9 20              CMP #$20        NÄCHSTE SEITE, WENN SPACE
```

```
12E7  F0 B5        BEQ $129E
12E9  4C 6D 1C     JMP RETURN TO KIM
```

ALPHA-SORT

E: This very flexible program allows to arrange data records of fixed length to be arranged in ascending or descending order. It is designed to work on ASCII-coded items but modifications are possible. The programmer may presribe what data field is selected for the sort, how many characters within the field are to be compared and what character terminates the comparision. Fields within the record may have variable length if they are separated by control characters like CR, LF, blank etc.

Das Sortieren von Informationen ist eine der häufigsten Tätigkeiten in der Datenverarbeitung. In technischen Anwendungen sollen Meßwerte nach Nummer der Meßstelle und/oder Zeit oder nach Meßwertgröße oder Größenklasse (Häufigkeiten) sortiert werden. Auch in den kaufmännischen Anwendungen werden 'Meßwerte' nach Größen klassifiziert: Kundenliste nach Umsatzgröße, Rennliste der Umsätze von Reisenden und gleiche Überlegungen für den mengenmäßigen Absatz nach Artikeln. Im Kaufmännischen werden ferner alle möglichen Arten von Bestandslisten geführt: Kunden nach Kundennummern sortiert, alphabetisch nach Namen, nach Ortsnamen, nach Postleitzahlgebiet, nach Vertreterbezirk, Land etc., ferner Bestandsverzeichnisse für Artikel.

Das Anlagen solcher Listen von Hand gehört zu den wirklich geisttötenden und fehleranfälligen Tätigkeiten. Betriebe mit eigenem Computer oder angeschlossenem Servicebüro lassen daher solche Aufgaben längst vom Rechner erledigen. Wer einen 65xx hat, braucht sich bald auch nicht mehr viel um solche Probleme zu kümmern, dieses Programm enthält eine vielseitig verwendbare und für die Bedürfnisse ausbaubare Lösung.

Die vorgenannten Anwendungen machen deutlich: Der Sortierbegriff steckt keineswegs immer in den ersten Bytes der Information. Nehmen wir eine Anschrift wie folgt:

```
Frau
Agatha Christie
Oscar-von-Miller-Ring 64 1/2
6800 Mannheim 1.
```

Der Familienname steht in der zweiten Zeile als zweites Item, die Postleitzahl mit 1-4 Zeichen befindet sich in der 4. Zeile, nach rechts durch einen Zwischenraum (blank) begrenzt. Jeder Bestandteil einer Anschrift, die man in der EDV als einen Datensatz bezeichnet, stellt ein 'Feld' dar, hier also Anrede, Vorname, Familienname, Straße, Haus-Nr., PLZ, Ort.

Anschriftendateien werden durchaus unterschiedlich aufgemacht. Die einzelnen Felder können durch einen Feldtrenner begonnen oder abgeschlossen werden, der ein auf Bildschirm oder Drucker nicht abbildbares/druckbares Zeichen darstellt. Zu diesen Steuerzeichen zählen insbesondere Wagenrücklauf (CR) und Zeilentransport (LF). Durch Zählen dieser Zeichen, insbesondere des LF, kann der Rechner feststellen, welche Zeile im Moment geprüft wird. Etwas schwieriger ist die Ansteuerung der Felder innerhalb einer Zeile. Möglicherweise sind z.B. in der Namenszeile Titel vorangestellt, Doppelnamen mögen mit oder ohne Bindestrich geschrieben sein. Man unterscheide also Karl Heinz und Karl-Heinz. Die Zählung von Zwischenräumen führt also nicht zwangsläufig z.B. auf das Feld des Nachnamens. Man halte sich weiterhin vor Augen, daß die Felder speziell bei Anschriften eine variable Länge haben.

Anzustreben ist immer eine dichte Packung der Information. Innerhalb eines Datensatzes mit fester Länge (z.B. 200 Bytes) möchte man dabei für ein einzelnes Feld von Datensatz zu Datensatz möglichst mit unterschiedlicher Länge arbeiten können. Bei schwieriger Struktur der Daten wird man dann vermehrt Steuerzeichen einsetzen müssen.

Sehr viel mehr Speicher braucht man dagegen, wenn man die einzelnen Felder für alle vorkommenden Eventualitäten ausreichend breit bemißt und beispielsweise anordnet, daß der Familienname grundsätzlich in Byte 60 beginnt. Die Bequemlichkeit der Ansteuerung erkauft man sich dann durch mehr Speicheraufwand.

> *Diese Vorüberlegungen werden bereits deutlich genug gemacht haben, daß es für das Sortierproblem keine allgemein gültige Lösung geben kann. Was zu tun ist, hängt immer vom Einzelfall ab.*

Wenn eine Datei erst noch anzulegen ist, kann man sie auf ein vorhandenes Sortierprogramm ausrichten. Ist dagegen eine konsequent gegliederte große Datei bereits vorhanden oder fällt sie aus den Meßwertaufnehmern in einer definierten Form an, so ist es im allgemeinen bequemer, die Gegebenheiten nicht zu ändern und stattdessen das Sortierprogramm entsprechend einzurichten.

ALPHA-SORT ist primär auf ASCII-kodierte Dateien ausgerichtet, also auf Meßwerte, die in diesem Code anfallen und auf kaufmännische Dateien. Der aufgezeigte Lösungsweg ist aber breit angelegt, eine Anpassung auf andere Bedarfsfälle kann ohne Schwierigkeiten erfolgen. Und wir werden noch einen HEXA-SORT für andere Codierung darstellen.

Eine Sortierung darf nicht nur auf eine Steile (1 Byte) beschränkt sein. Reines Zahlenordnen wäre trivial. Die Zahl der zu sortierenden Stellen muß vorgebbar sein, ebenso muß man das zu sortierende Feld auswählen können. Zu sortierende Information steht nicht allein, sie steht in Zusammenhang mit der sonstigen Information des Datensatzes. Wenn man also Anschriften nach Postleitzahlen sortiert, möchte man nicht nur die Kundenzahl erfassen, sondern auch die dazu gehörenden Kundennamen und Umsätze.

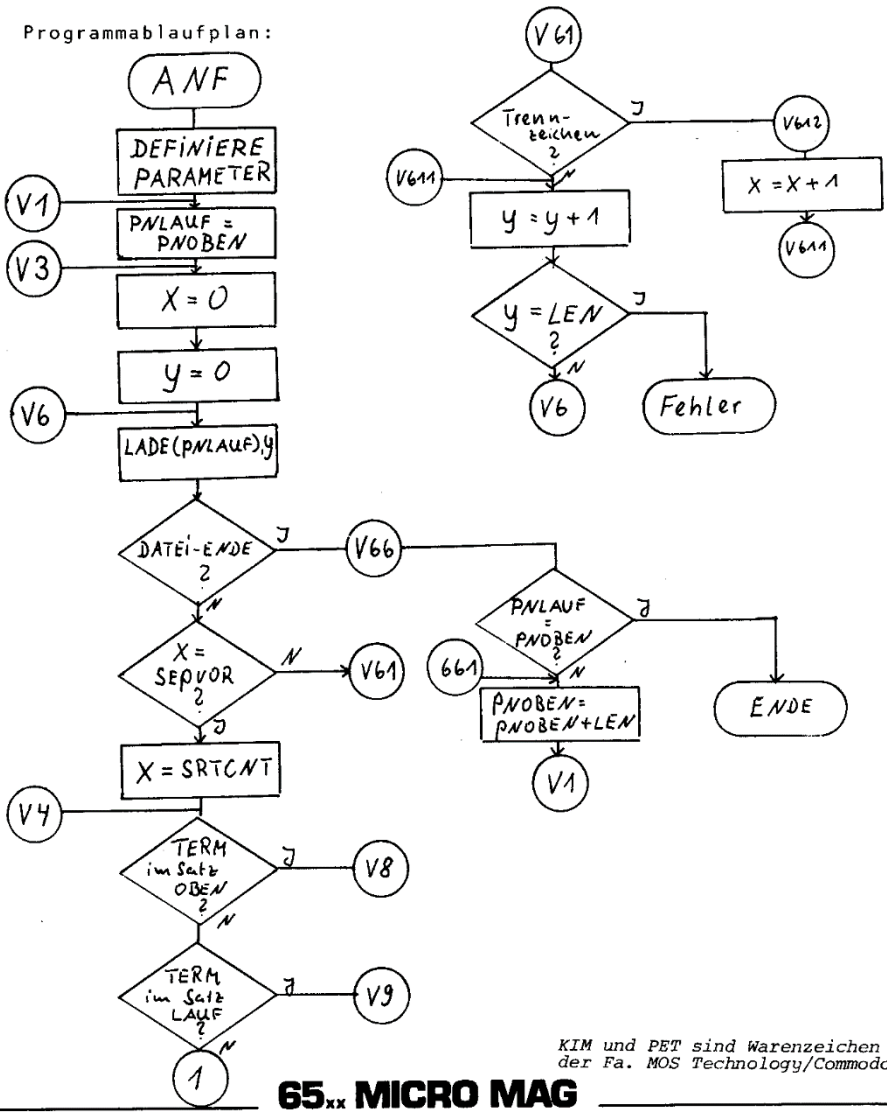
Weiterhin möchte man nach aufsteigendem oder nach absteigendem Wert des Sortierbegriffes anordnen können, wobei alphabetischen Zeichen auf Grund ihrer Kodierung ebenso ein Wert zukommt, wie Ziffern.

Man studiere dazu die Tabelle der ASCII-Kodierung. Beim 7-Bit-ASCII liegen die druckbaren Werte zwischen 20 (hex) und 7E, beginnend mit Sonderzeichen, Ziffern, wiederum Sonderzeichen, Großbuchstaben und Kleinbuchstaben. Zwischen 00 und 1F liegen Steuerzeichen. Für das auf- oder absteigende Sortieren bedeutet dieser Code, daß es eine Gleichwertigkeit zwischen Groß und Kleinbuchstaben nicht gibt. Es stünde also 1ALPHA vor AACHEN oder AA bis ZZ.. ebenso vor Aachen. Eine Gleichwertigkeit von Groß- und Kleinbuchstaben erreicht man, indem man das unterscheidende Bit 5 maskiert.

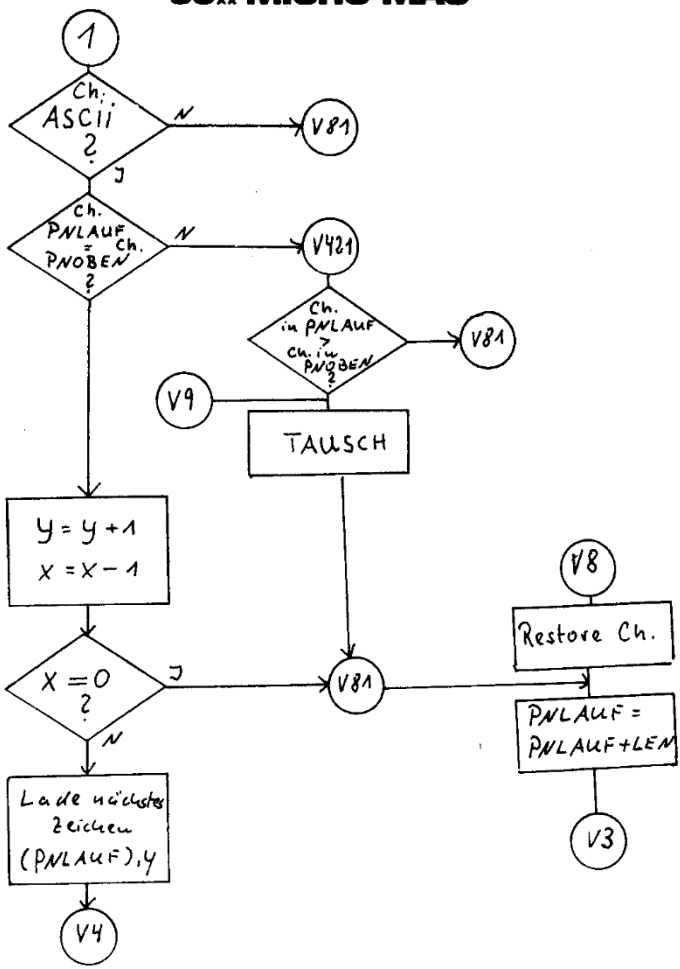
Besonders als Illustration für den 'Leitfaden für die Programmierung' folgt zunächst der Programmablaufplan, der für sich spricht. Das Lösungsverfahren entspricht dem Dreieckstausch (bubble up). Man vergleicht jeweils den vordersten mit dem laufenden Satz. Ist jener kleiner, so wird er nach oben getauscht. Nach dem ersten Überstreichen der Sätze steht daher derjenige mit dem niedrigsten Sortierbegriff vornan, nach dem zweiten Durchgang steht der Satz mit zweitniedrigstem Sortiermerkmal an zweiter Stelle usw. Nach jedem Durchgang wird die Adresse des vordersten Satzes um eine Satzlänge (LEN) erhöht.

Eine vom Programmierer definierbare Anzahl (SEPVOR) von Feldtrennzeichen, deren Code in SEP1 und SEP2 niedergelegt wird (wie z.B. für CR und LF) muß zunächst überstrichen sein, ehe der eigentliche Sortiergang ansetzt. Dies dient der Ansteuerung des Sortierfeldes. Innerhalb des Sortierfeldes wird der Vergleich abgeschlossen, wenn die zu sortierende Stellenzahl SRTCNT erreicht ist oder wenn vorher ein frei definierbarer Sortierbegrenzer TERM auftritt, z.B. ein Zwischenraum. Unplanmäßig codierte Datensätze, Krücken, bilden den leicht erkennbaren Vor- oder Nachspann der Hauptsortierung.

Programmablaufplan:



KIM und PET sind Warenzeichen der Fa. MOS Technology/Commodore



ALPHA-SORT

IN DER ZEROPAGE ZU ÜBERGEBENDE PARAMETER:

00D1	PNOBEN	.BYTE	POINTERADRESSE FÜR DEN JEWEILS
00D2		.BYTE	VORDERSTEN DATENSATZ
00D3	LEN	.BYTE	SATZLÄNGE $BYTES
00D4	SEP1	.BYTE	ASCII-CODE EINES FELDBEGRENZERS
00D5	SEP2	.BYTE	DITO
00D6	SEPVOR	.BYTE	$ZAHL DER VOR DEM SORTIERVORGANG ZU ÜBERSTREICHENDEN FELDBEGRENZER SEP1 ODER SEP2
00D7	TERM	.BYTE	FELDBEGRENZER FÜR VERGLEICHSFELD
00D8	SRTCNT	.BYTE	ZU SORTIERENDE $STELLENZAHL
00D9/DA	PNLAUF		POINTER PNLAUF

```
0200  D8          ANF   CLD
0201  A5 D1       V1    LDA PNOBEN      PNLAUF = PNOBEN
0203  85 D9             STA PNLAUF
0205  A5 D2             LDA PNOBEN+1
0207  85 DA             STA PNLAUF+1

0209  A2 00       V3    LDX #$ 00       COUNTER FOR SEPARATORS
020B  A0 00             LDY #$ 00       INDIRECT ADDRESSER
020D  B1 D9       V6    LDA (PNLAUF),Y  GET FIRST BYTE OF NEW RECORD
020F  C9 1C             CMP #$ 1C       FILE SEPARATOR ? (ASCII)
0211  F0 27             BEQ V66         IF YES
0213  E4 D6             CPX SEPVOR      SCANNED ENOUGH?
0215  D0 3D             BNE V61         DO MORE SCANNING
0217  A6 D8             LDX SRTCNT      SET UP AS A COUNTER
0219  48        V4    PHA             SAVE CHARACTER ON STACK
021A  B1 D1             LDA (PNOBEN),Y  CHARACTER FROM FOREMOST RECORD
021C  C5 D7             CMP TERM        FIELD ALREADY DONE ?
021E  F0 4B             BEQ V8          ADVANCE TO NEXT RECORD
0220  68                PLA             RESTORE CHARACTER
0221  C5 D7             CMP TERM        THIS FIELD TERMINATED?
0223  F0 54             BEQ V9          CHANGE CONTENTS OF RECORDS

0225  C9 20       V41   CMP #$ 20       PRINTABLE  ASCII CHARACTER?
0227  90 43             BCC V81         IF BELOW LIMIT
0229  C9 7F             CMP #$ 7F       UPPER LIMIT
022B  B0 3F             BCS V81         IF BEYOND

022D  D1 D1             CMP (PNOBEN),Y  COMPARE TO CORRESPONDENT CHAR.
022F  D0 5A             BNE V421        CHECK FURTHER
0231  C8                INY             FOR NEXT ADDRESSING
0232  CA                DEX             ONE LESS CHAR. TO BE COMPARED
0233  F0 37             BEQ V81         GET NEXT RECORD
0235  B1 D9             LDA (PNLAUF),Y  GET NEXT CHAR. FROM CURRENT REC.
0237  4C 19 02          JMP             COMPARE MORE CHARACTERS

023A  A5 D2       V66   LDA PNOBEN+1    ARE POINTERS EQUAL?
023C  C5 DA             CMP PNLAUF+1
023E  D0 09             BNE V661        IF NOT
0240  A5 D1             LDA PNOBEN      SAME FOR LOW
0242  C5 D9             CMP PNLAUF
0244  D0 03             BNE V661
0246  4C 4F 1C          JMP START       RETURN TO MONITOR, ALL DONE.

0249  18        V661  CLC             PREPARE FOR ADD
024A  65 D3             ADC LEN         PNOBEN = PNOBEN + LEN
024C  85 D1             STA PNOBEN
024E  90 B1             BCC V1          ON NO CARRY
0250  E6 D2             INC PNOBEN+1    ADJUST  HI ADDRESS
0252  B0 AD             BCS V1          BRANCH ALWAYS

0254  C5 D4       V61   CMP SEP1        ONE OR OTHER SEPARATOR?
0256  F0 0F             BEQ V612        INCREMENT COUNT
0258  C5 D5             CMP SEP2
025A  F0 0B             BEQ V612

025C  C8          V611  INY             FOR NEXT ADDRESSING
025D  C4 D3             CPY LEN         DON'T TOUCH NEXT RECORD!
025F  D0 03             BNE             SKIP TO *+03
0261  4C 29 19          JMP LOADT9      ERROR EXIT, SHOW FF FF 1C
```

```
0264  4C 0D 02          JMP V6          COMPARE NEXT CHARACTER

0267  E8        V612    INX             SEPARATOR IS COUNTED
0268  4C 5C 02          JMP V611

026B  68        V8      PLA             ADJUST STACK POINTER
026C  18        V81     CLC
026D  A5 D9             LDA PNLAUF      PNLAUF = PNLAUF+LEN
026F  65 D3             ADC LEN
0271  85 D9             STA PNLAUF
0273  90 94             BCC V3
0275  E6 DA             INC PNLAUF+1
0277  B0 90             BCS V3          BRANCH ALWAYS

0279  A4 D3     V9      LDY LEN         ONE FULL RECORD TO BE EXCHANGED
027B  88        V91     DEY             ADJUST FOR COUNT AND ADDRESSING
027C  B1 D1             LDA (PNOBEN),Y
027E  AA                TAX             SAVE CHARACTER IN X
027F  B1 D9             LDA (PNLAUF),Y
0281  91 D1             STA (PNOBEN),Y
0283  8A                TXA             RESTORE CHARACTER
0284  91 D9             STA (PNLAUF),Y
0286  98                TYA             GET STATUS BIT FROM Y
0287  D0 F2             BNE V91
0289  F0 E1             BEQ V81         BRANCH ALWAYS

028B  B0 DF     V421    BCS V81         INSERT 90 DF FOR DESCENDING ORDER
028D  90 EA             BCC V9          INSERT B0 EA FOR DESCENDING ORDER
                                        OF SORT
028F  EA                NOP             RELOCATION BYTE FOR RALOAD
```

Zur Anwendung dieses eigentlich sehr kurzen und dennoch leistungsfähigen Programmes seien noch folgende Hinweise gegeben. Mit den Instruktionen in V421 wird gesteuert, ob die Datensätze in aufsteigender oder in absteigender Folge des Sortierbegriffes angeordnet werden sollen.

Immer ist vorausgesetzt, daß sich alle zu sortierenden Datensätze resident im Speicher befinden und daß unmittelbar auf den letzten Datensatz der FILE SEPARATOR '1C' folgt. Für das Sortieren sehr großer Dateien wird man externe Speichermedien benötigen wie Magnetband oder Diskette. Man legt Puffer für zwei möglichst große Blöcke an, die jeweils eine Vielzahl von Datensätzen enthalten. Pro Sortiergang im Speicher kann man dann jeweils erreichen, daß alle Datensätze des ersten Blockes ein niedrigeres Sortiermerkmal als die des zweiten Blockes haben. Den ersten Block bringt man auf magnetischen Zwischenspeicher, bringt den zweiten an den residenten Speicherplatz des ersten und liest einen dritten Block ein, um ihn gegen den zweiten zu sortieren usw. Es gibt Formeln, wie oft man einlesen, sortieren und zwischenspeichern muß, wenn eine Datei X Datensätze enthält, die jeweils zu Y Stück in einem Block zusammengefaßt sind. Bei solchen großen Dateien wird es auf schnelle periphere Speichermedien ankommen, im Prinzip aber leistet vorstehendes Programm das gleiche wie der Sort eines großen Gerätes. *R.L.*

****

KLEINANZEIGEN DER LESER:

SUCHE VERSIERTEN BASIC-PROGRAMMIERER IM POSTLEITZAHLRAUM 48.
I. DOHMANNN, ELEKTRONISCHE BAUGRUPPEN, IM WIEHAGEN 13-15, 4830 GÜTERSLOH 12, TEL.: 05241 - 6 74 80.

MARKTFORSCHUNG - LESERBEFRAGUNG

Sehr geehrter Leser!

Diese Zeitschrift wird für Sie geschrieben und gestaltet. Sie wurden ständiger Leser, weil die gebotene Information für Sie nützlich ist. Gleichwohl sind Ihre Wünsche und auch Ihre Beurteilung hier noch zu wenig bekannt.

Der Herausgeber hat daher einen kleinen Fragebogen vorbereitet und diesem Heft beigelegt. Er bittet Sie höflich um die kleine Mühe der Beantwortung und um möglichst baldige Rücksendung. Ihre Antworten dienen der weiteren Gestaltung, bleiben anonym und werden an keine dritte Person weitergegeben.

Wie Sie wissen, ist das 65xx MICRO MAG bemüht, vor allem nützliche Software zu vermitteln. Es bringt dabei Erstveröffentlichungen und stellt sie möglichst in einen Zusammenhang. Bei der Kommentierung wird versucht, für den Newcomer ausreichende Erklärungen zu geben und den erfahrenen Leser nicht mit Trivialitäten zu langweilen. In der Summe ist eine Zeitschrift entstanden, die sich von anderen Veröffentlichungen nicht nur durch die Fülle der Information für 65xx abhebt, sondern auch durch eine Darstellungsart, die dem anspruchsvollen Leser verpflichtet ist. Wie beurteilen Sie diese Linie?

Dem 65xx MICRO MAG stehen nicht viele und große Werbemöglichkeiten offen. Wenn diese Zeitschrift auch einen sehr erfreulichen Aufschwung nimmt, so muß sie doch um eine größere Leserschaft bemüht bleiben. Unterstützen Sie bitte die Leistungskraft der Zeitschrift durch persönliche Weiterempfehlung und auch durch gelegentliche Beiträge zum Inhalt.

Mit den besten Wünschen zum Jahreswechsel Roland Löhr.

****

MICRO 7, S. 5, Rick Auricchio

Breaker, An APPLE II Debugging Aid

Debugger-Programm mit Listing und Kommentaren, um Breakpoints zu setzen und zu löschen, ohne den Maschinencode zu zerstören. Anzeige der Register.

MICRO 7, S. 19, J.S. Green

650x Opcode Sequence Matcher

Dienstprogramm, das ein Anwenderprogramm auf gleiche Instruktionsfolgen abrastert, um Wiederholungen, Redundanzen, zu erkennen.

MICRO 7, S. 49, Dr. Barry Tepperman

Kimbase

6-ziffrige Ganzzahlen der Zahlenbasen 2-16 (dez.) können beliebig ineinander umgerechnet werden. Ausführlich kommentiertes Programm.

BYTE 10/78, S. 34, K.-M. Chung und H. Yuen

A 'Tiny' PASCAL Compiler, Part 2

Compiler zur Erzeugung des intermediären P-Codes aus PASCAL-Statements. Programmlisting in BASIC.

BYTE 11/78, S. 182, K.-M. Chung und H. Yuen

A 'Tiny' PASCAL Compiler, Part 3

Abschluß der Serie. Bewertung von Compiler-Konzepten. Beschreibung des Translators für P-Code zu 8080-Maschinencode. Ohne Listing.

KILOBAUD 10/78, S. 26, Dr. J. Tageser

Budget System with KIM

Zusammenspiel des TVT6-Videomonitors mit einem ASCII-Keyboard als KIM-Erweiterung. Flowcharts und Listings für das Betriebssystem AKIM. Die Speicherbeengung durch TVT6 wird umgangen.

GWK EURO BOARD EXPANSION SYSTEM für KIM

Es bietet sowohl dem professionellen Anwender,als auch dem Hobbyisten,die Möglichkeit,seinen KIM optimal zu erweitern. Dies wird erreicht durch die folgenden grundlegenden Konzepte: Doppelte Pufferung von Daten- und Adressbus bewirken größtmögliche Störsicherheit.

Die Adressdecodierung der einzelnen Komponenten erfolgt auf jeder Karte.Das heißt,jede Komponente kann beliebig adressiert werden.Es wird kein Adressraum verschenkt.

Der GWK EURO BOARD EXPANSION SYSTEM Bus und die Steckverbinder der einzelnen Komponenten sind pinkompatibel zum Expansion Connector des KIM.Daraus ergibt sich,daß einzelne Karten auch ohne das MOTHER BOARD direkt am KIM laufen.

Bisher sind folgende Komponenten lieferbar.

MOTHER BOARD ohne Steckerleisten 325,--

Einbau in Gehäuse 19 Zoll oder direkt an KIM steckbar.Daten- Adress- und Control Bus sind voll gepuffert und dynamisch abgeschlossen.9 Steckplätze für Expansion,3 für Application, 1 mit genügend Platz für ein Netzteil.

RAM BOARD 895.--

12 K Byte statisches Ram.In Blöcken zu je 4K Byte beliebig adressierbar.

EPROM BOARD ohne Eproms 395,--

12K Byte für die 1K Eproms 2708 oder 2758.

EPROM PROGRAMMER BOARD 745,-- 795,--

Mit residentem Betriebsprogramm.Softwareumschaltung für 1K-,2K-,4K Byte EPROMS xx08/58,xx16,xx32.Zwei Ausführungen: Für Eproms mit einer oder mit drei Betriebsspannungen.

CROSS ASSEMBLER 650X - PDP11 985,--

Zusätzliches Info anfordern.

EPROM TMS 2708,1K Byte,3 Vers.Spng.	23,--
EPROM TMS 2516,2K Byte,1 Versorg.Spng.	115,--
STECKERLEISTEN zum MOTHERBOARD	
22 Polig für Netzteil Amphenol 143-022-01-102	7,12
44 Polig für Exp.u.Appl. " 225-22221-110	9,60

Alle Preise zuzüglich Versandspesen und MWST.

GWK TECHNISCHE ELEKTRONIK
HARDWARE SOFTWARE SYSTEMENTWICKLUNG
Aachener Str.56 · D 5132 Uebach Palenberg
Tel.: 02451 / 41911

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

HERAUSGEBER:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
2070 AHRENSBURG
☎ (04102) 55 816

65xx MICRO MAG erscheint zweimonatlich als Manuskriptdruck. Beiträge, die nicht besonders gekennzeichnet sind, stammen vom Herausgeber, der als Datenverarbeitungsfachmann auch freiberuflich beratend tätig ist.

BEZUGSBEDINGUNGEN: Abonnement für 6 Ausgaben im Inland DM 40,- (Endpreis). Ausland: DM 46,- (surface mail).
Firmen erhalten Rechnung. Rechnungserteilung sonst nur auf Wunsch.
Richten Sie bitte Ihre Überweisung/Euroscheck an:

Roland Löhr, Konto 20/01121, Vereins- und Westbank, BLZ 200 300 00.

Wegen des Zusammenhanges der Hefte erhalten hinzukommende Abonnenten, wenn nicht anders gewünscht, Lieferung ab erstem Heft mit Laufzeit von da an. Einzelne Hefte können zu DM 7,- inkl. Porto nachbezogen werden.

Informationsblätter für Werbetreibende und Distributoren stehen zur Verfügung.

Beiträge aus dem Leserkreis sind sehr willkommen. Hinweise dazu in Heft 2.

REDAKTIONS- UND ANZEIGENSCHLUSS FÜR NR. 5 IST DER 2. FEB. 1979

In den nächsten Heften finden Sie u.a.:
HEXA-SORT - Ein Leitfaden für die Programmierung, Fortsetzung mit Beispielen - Fortsetzungen für 65xx-Makros, für Advanced Subroutine Package - Ausführliche Berichte über den AIM 65 und SYM-1 -

MICRO 7, S. 25, Michael McCann

A Memory Test Program for the Commodore PET

Programm in BASIC zur Speicherprüfung des PET

65xx MICRO MAG

KILOBAUD 10/78, S. 110, Philip Ngai

Build a One-Chip Single-Stepper

Benutzung eines Flip-Flops, um bei 65xx-Selbs-bausystemen Einzelschritt-Befehlsausführung zu ermöglichen.

65xx MICRO MAG

MICRO EXTRA 6/78, S. 8, Edgar Gassner

Erfahrungsbericht des PET 2001

Bewertung des Gerätes, Darstellung kleiner nützlicher Routinen in BASIC zur Programmeingabe in Assembler. (Feltron Publikation)

65xx MICRO MAG

KILOBAUD 11/78, S. 67, Easton Beymer

Universal Number Converter

Programm in BASIC zur Umwandlung von Zahlen zur Basis 2-16 (dez.) beliebig untereinander.

65xx MICRO MAG